Introduction to Deep Neural Network

Prof. Chia-Yu Lin

Yuan Ze University

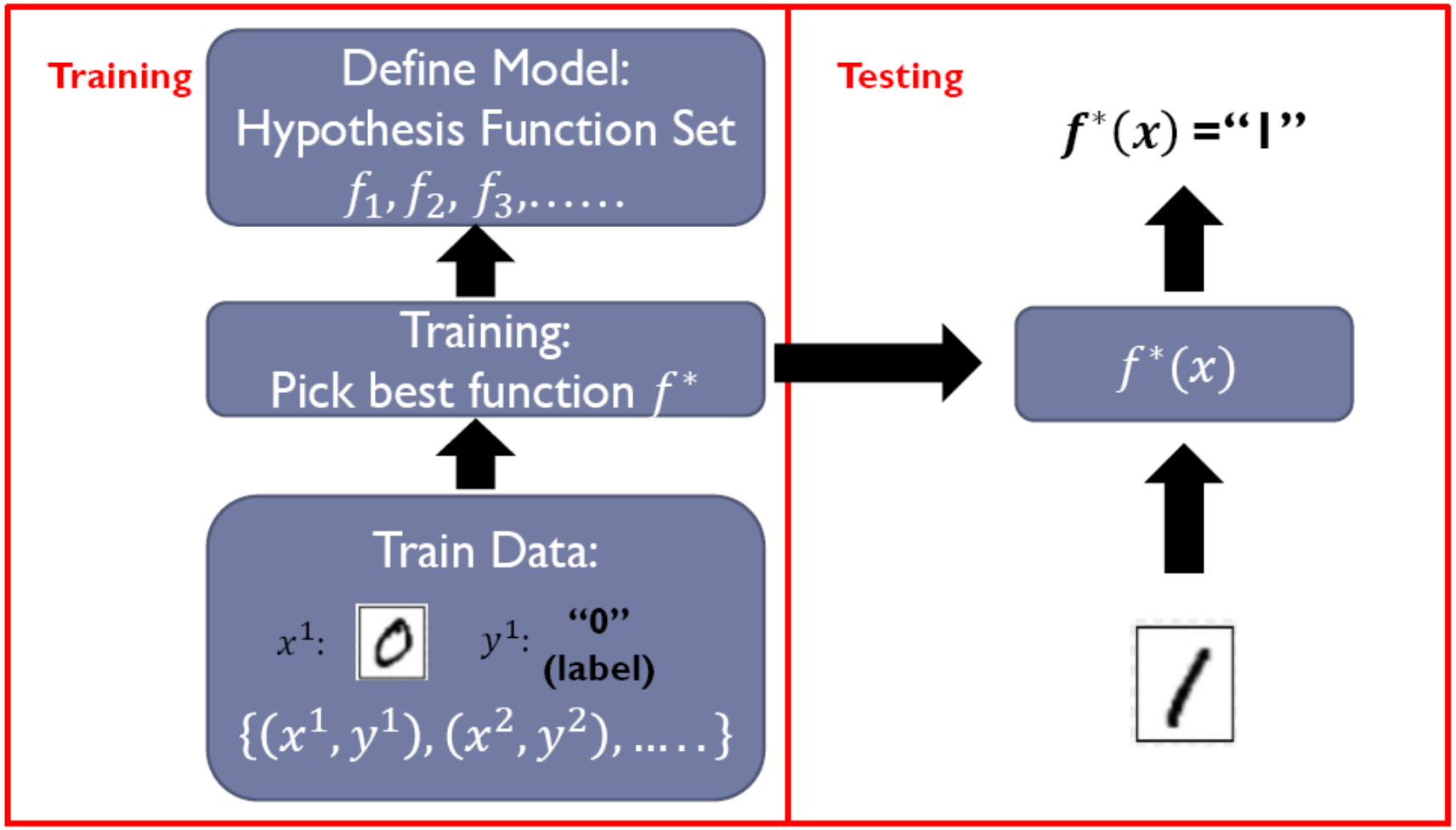
2021 Spring

Thanks to the slides of Prof. H.-T. Lin and Prof. Lee Hung-Yi Lee from NTU.

Outline

- Why Deep Learning
- Interesting Project Overview
- Training Model
  - Prepare dataset
  - Build model
  - Define loss
    - Mean Square
    - Cross-entropy
  - Optimization
    - Gradient decent
    - Increase the performance of optimization
      - Tips 1:Tune Learning Rate
      - Tips2: Stochastic Gradient Descent
      - Tips3: Feature Scaling
    - How to update many parameters
- Testing Model
- DNN Discussion

Learning

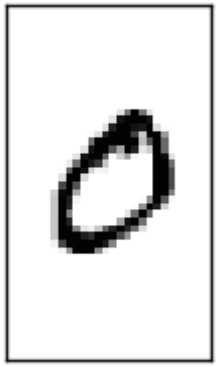

Training
Define Model:
Hypothesis Function Set
$f_1, f_2, f_3, \ldots\ldots$
Training:
Pick best function f^*
Train Data:
x^1:
y^1:
"0"
(label)
$\{(x^1, y^1), (x^2, y^2), \ldots\ldots\}$
Testing
$f^*(x)$ ="1"
$f^*(x)$

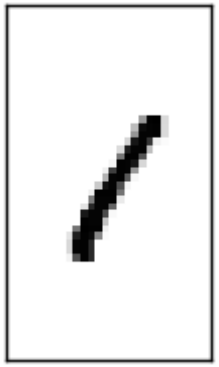
Why Deep Learning?

- In the past
  - We need feature engineering (take lots of human labor)
- Deep Learning
  - It learns suitable features for this task.

Example

- Detect if this image is “zero” or “one”





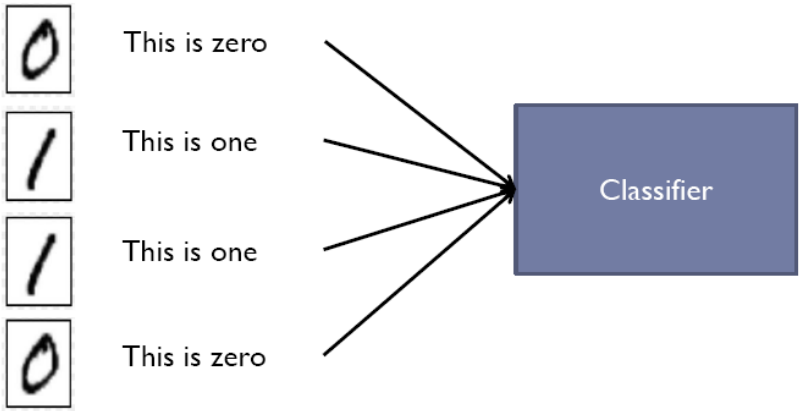
Traditional vs. Deep Learning

Traditional Method

Feature #1 = if black part is round
Feature #2 = ratio of black pixels in image
…..
…..
…..
Feature #N = center of image pixel is white

SVM
Logistic regression
…..
…..
…..

Deep Neural Network

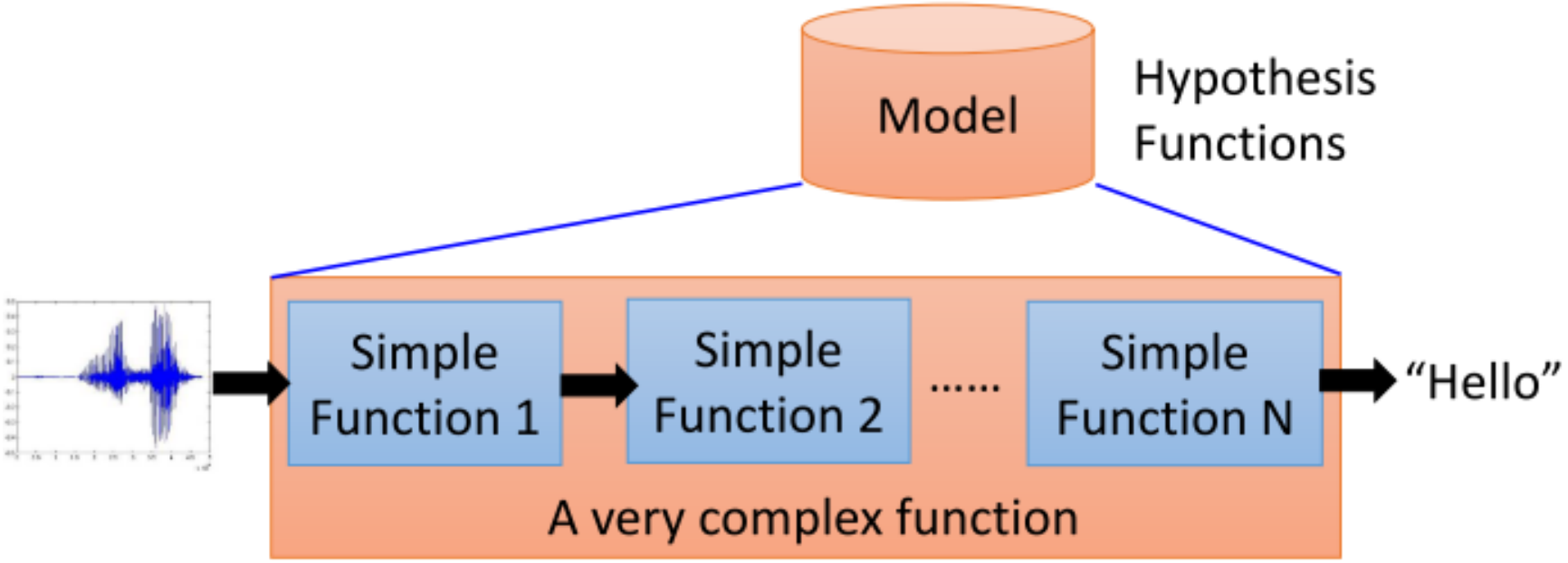


Human extract features =>Human error

Classifier would automatically learn features based on dataset

What is Deep Learning?

- Production Line (生產線)



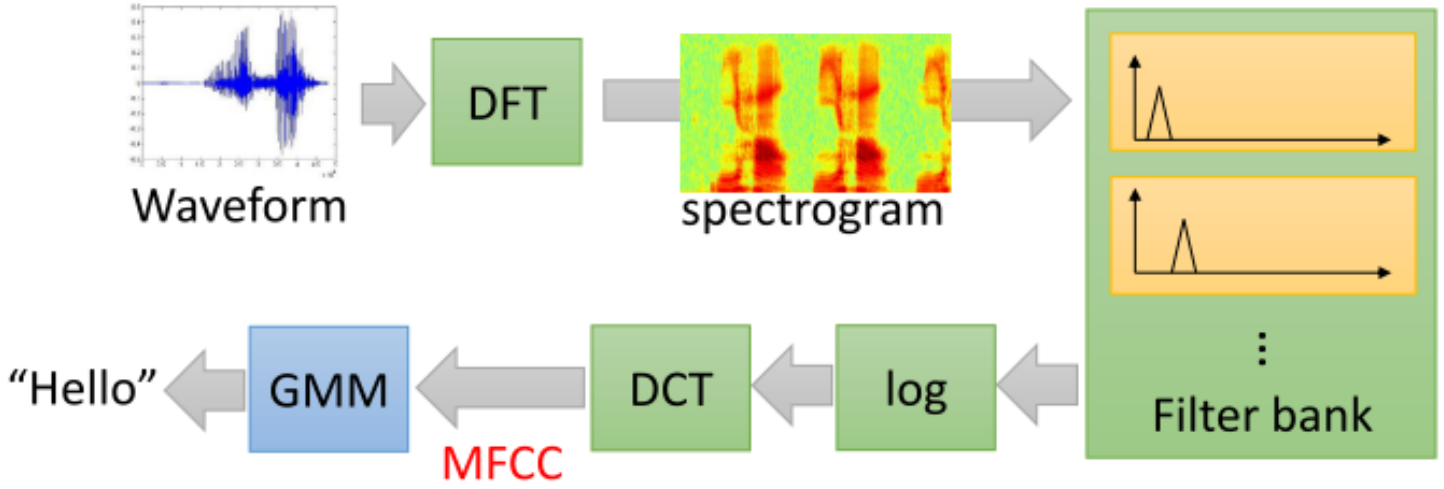
End-to-end training:

What each function should do is learned automatically

Shallow Approach

- Example: Speech Recognition
- 5 hand-crafted steps
- Only one step learns from data.

Shallow Approach

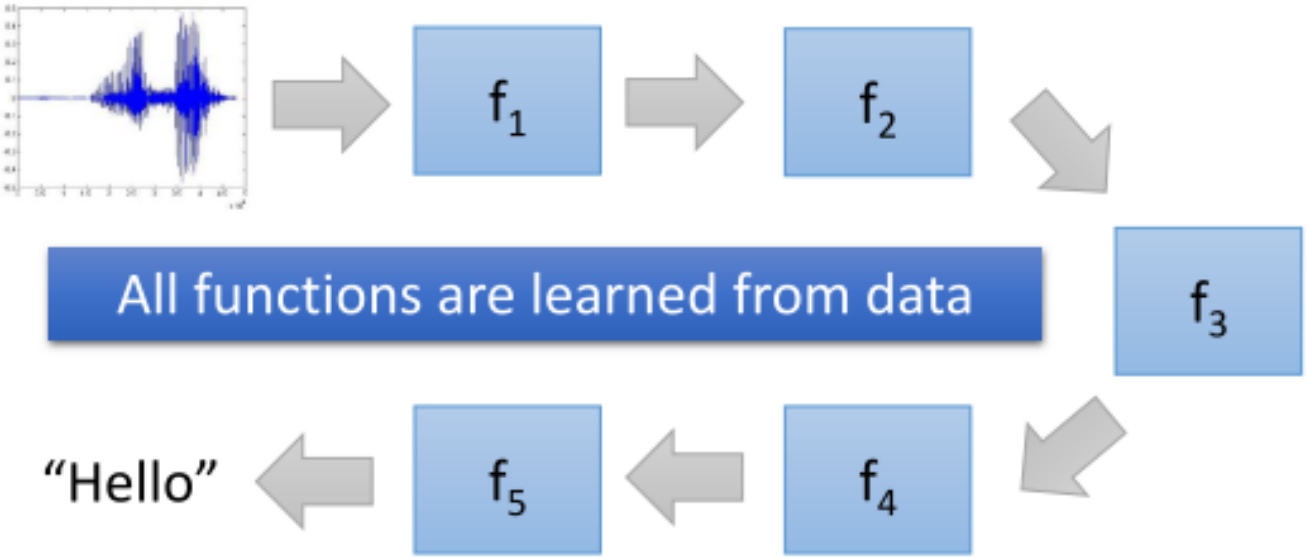


Each box is a simple function in the production line:

:hand-crafted :learned from data

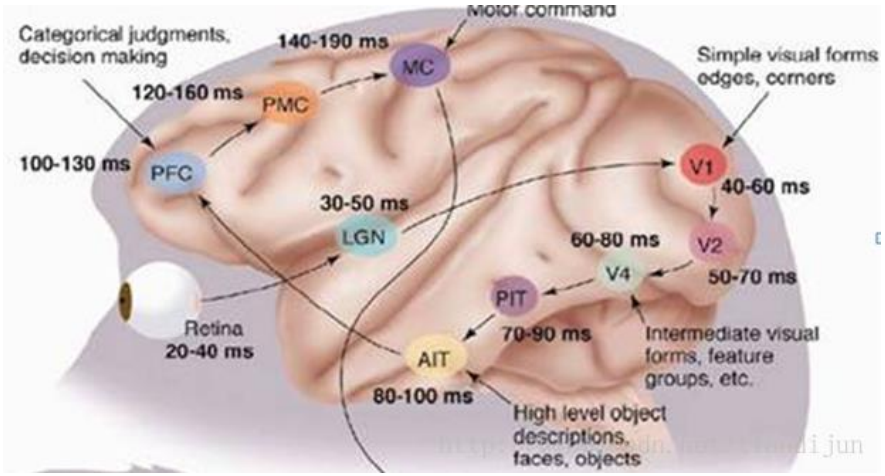
Deep Leaning

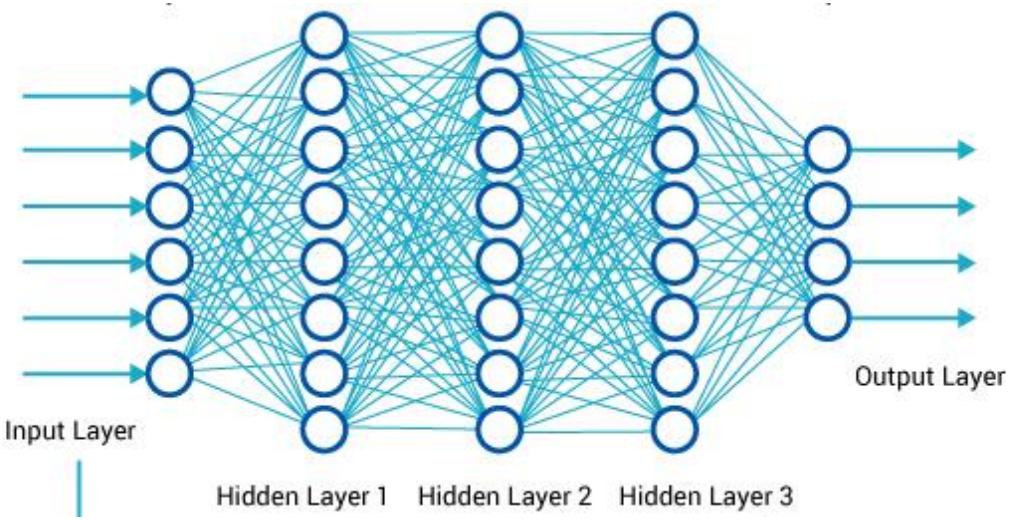
- Each function in the process can learn from data.



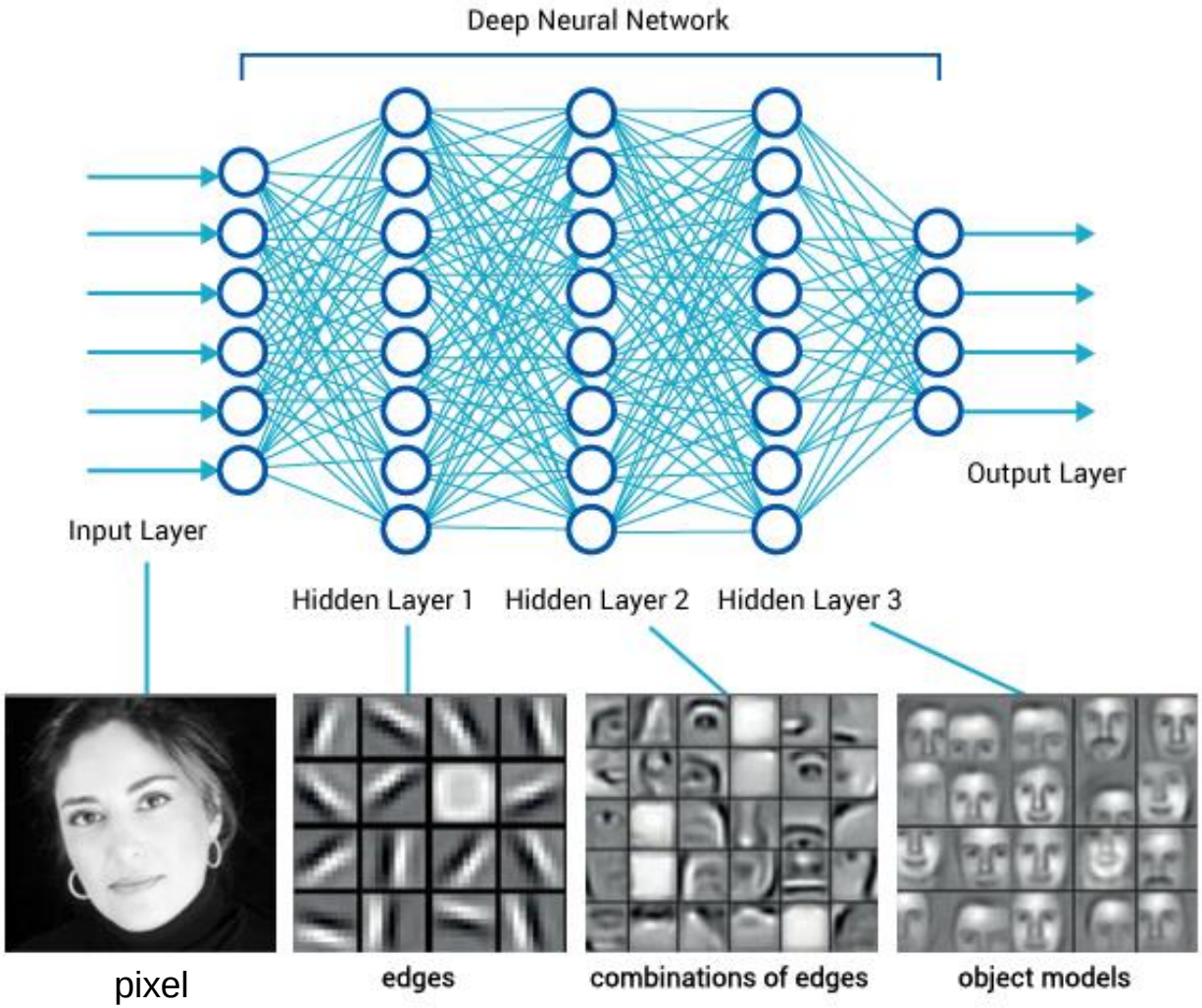
Less engineering labor, but machine learns more

Idea from human brain



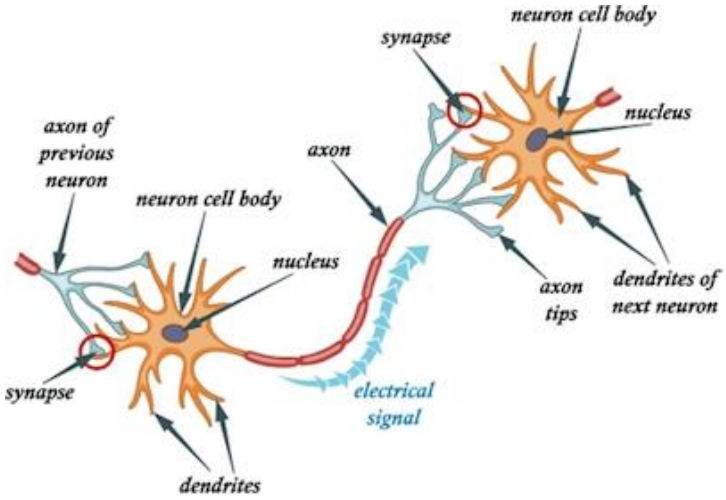


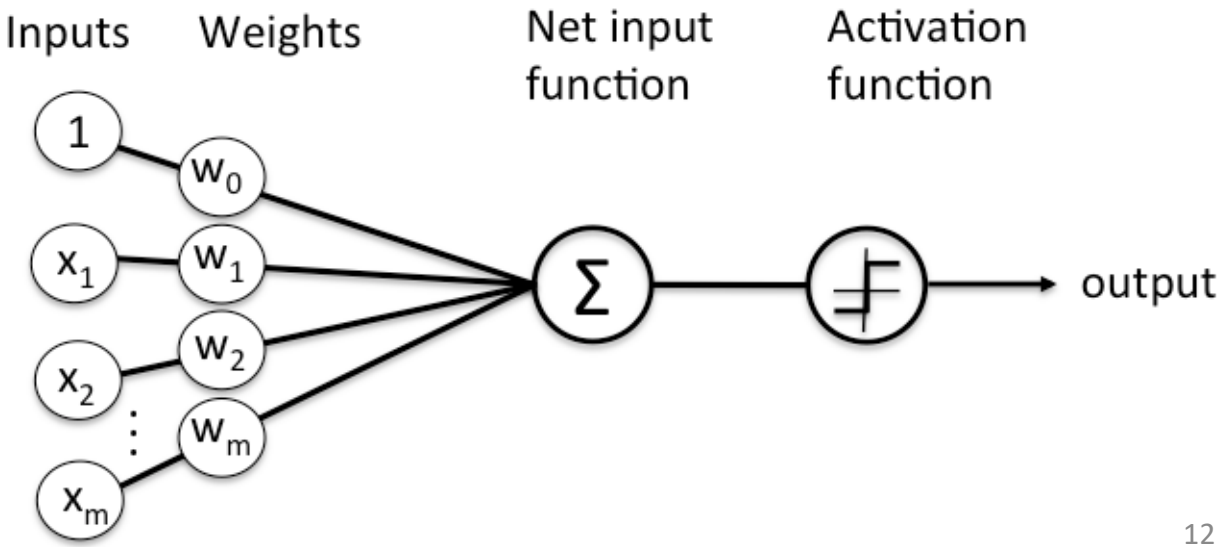
Artificial Neuron Network(ANN)



Source:http://www.amax.com/blog/?p=804

Neuron

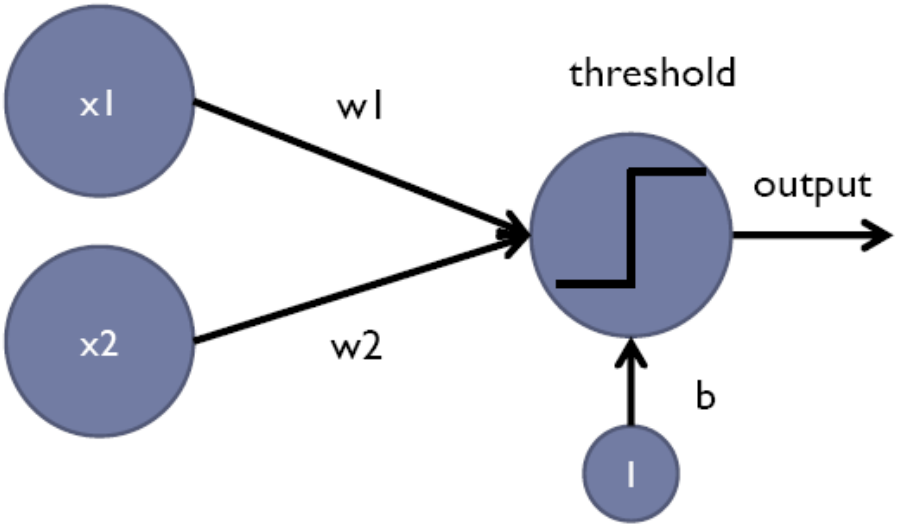

synapse
neuron cell body
axon of previous neuron
axon
nucleus
neuron cell body
nucleus
axon tips
dendrites of next neuron
synapse
electrical signal
dendrites


Inputs
Weights
Net input function
Activation function
1
x_1
x_2
x_m
w_0
w_1
w_2
w_m
Σ
output

Example (1/2)

- How to use the idea of neural network to simulate “NAND” gate?

NAND gate truth table		
x1	x2	output
0	0	1
0	1	1
1	0	1
1	1	0

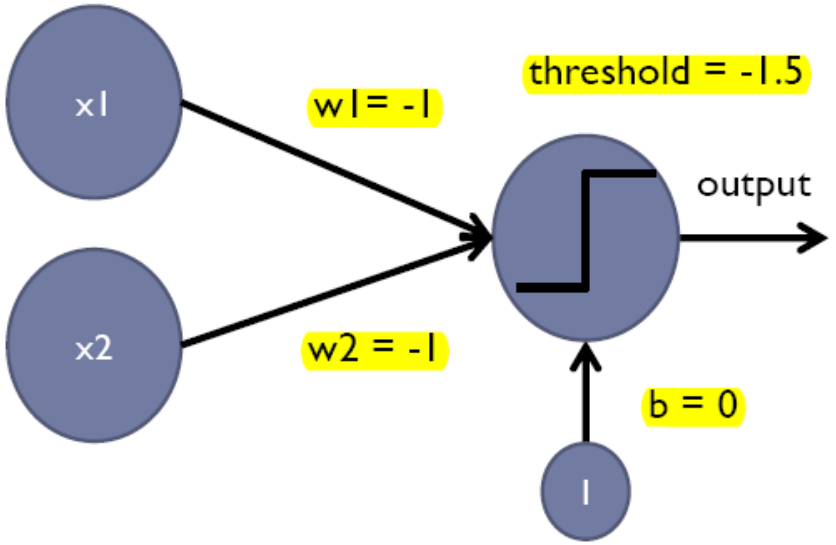


$$output = \begin{cases} 0 & if\ x1 * w1 + x2 * w2 + b \leq threshold \\ 1 & if\ x1 * w1 + x2 * w2 + b > threshold \end{cases}$$

Example (2/2)

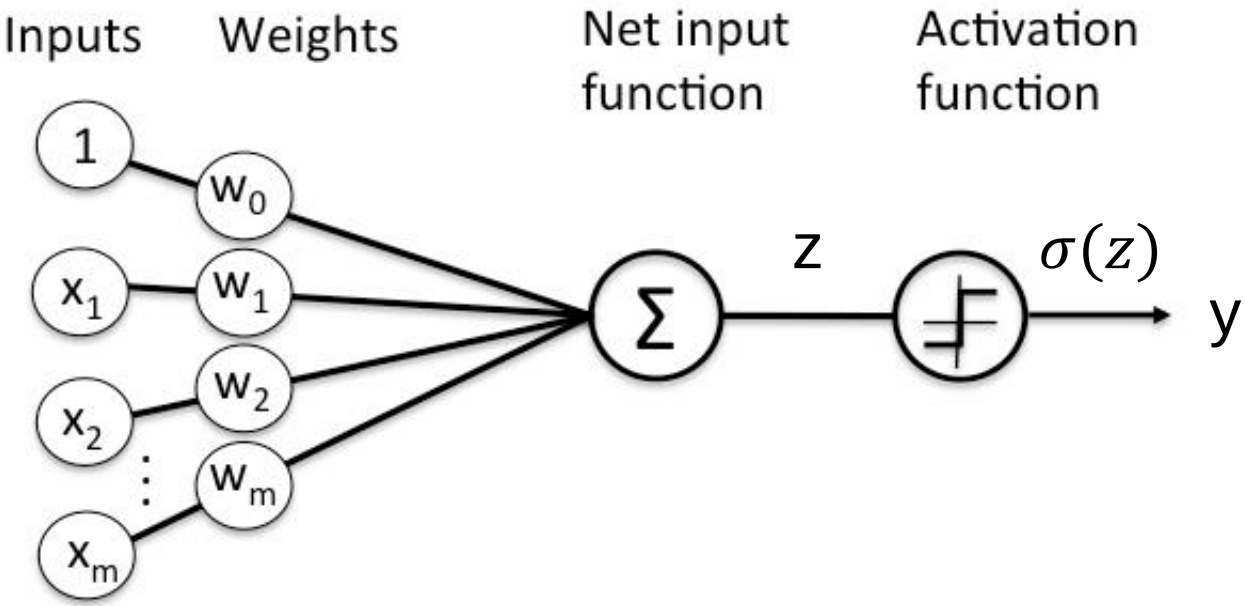
- Set weighting, bias and threshold.

NAND gate truth table		
x1	x2	output
0	0	1
0	1	1
1	0	1
1	1	0



$$output = \begin{cases} 0 & if\ x1 * w1 + x2 * w2 + b \leq threshold \\ 1 & if\ x1 * w1 + x2 * w2 + b > threshold \end{cases}$$

Neuron

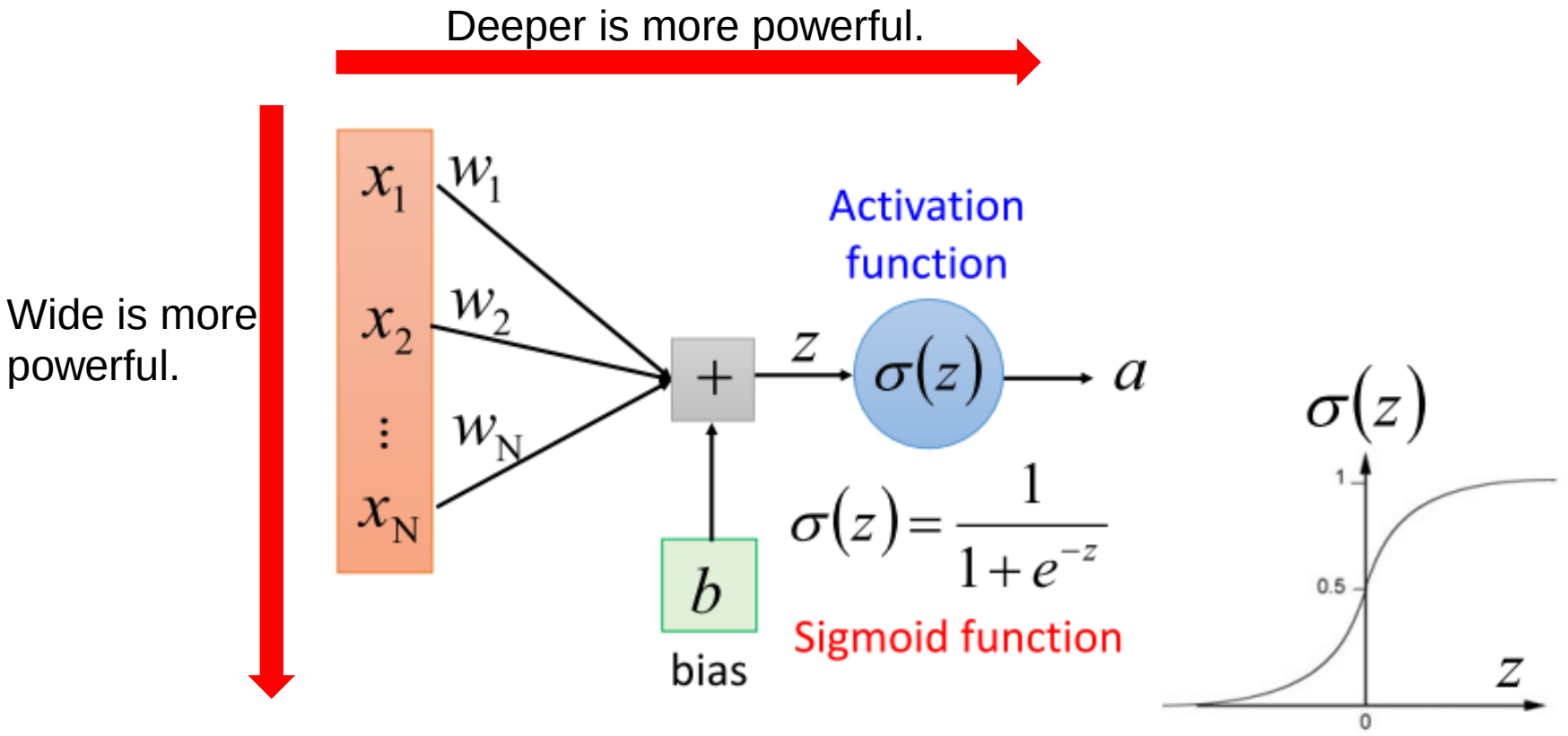


$z = 1*w_0 + x_1*w_1 + x_2*w_2 + \ldots + x_m*w_m$

$y = \sigma(z)$

A Neuron for Machine

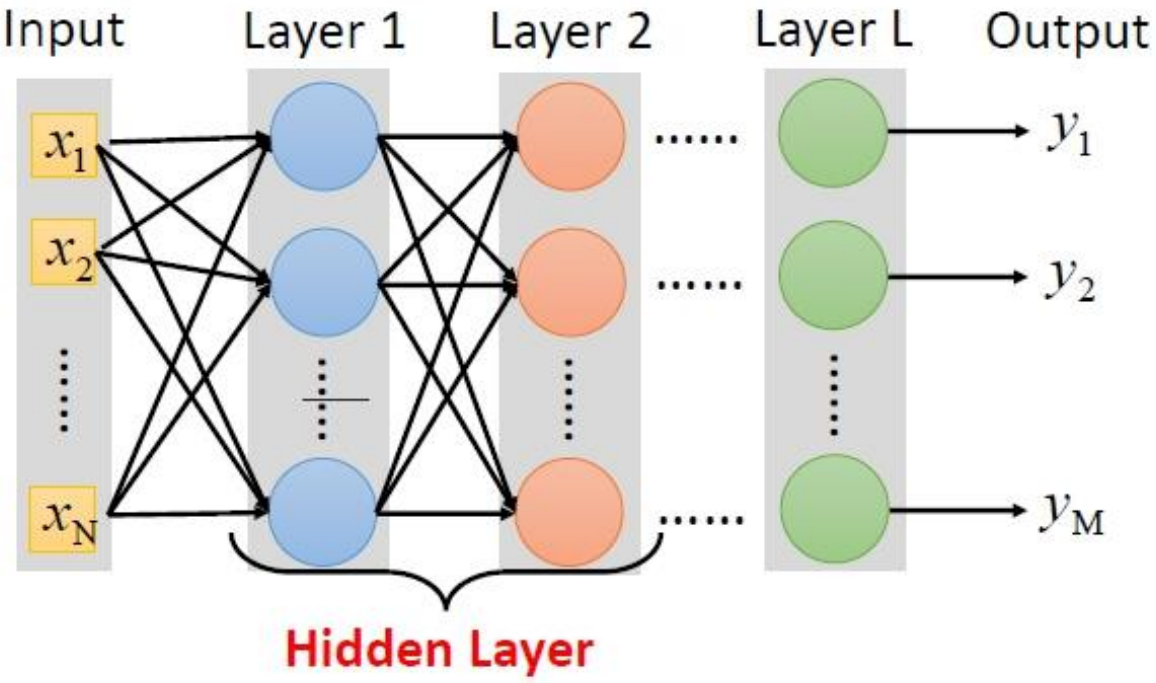
- Each neuron is a very simple function.



Weights and biases are called network parameters

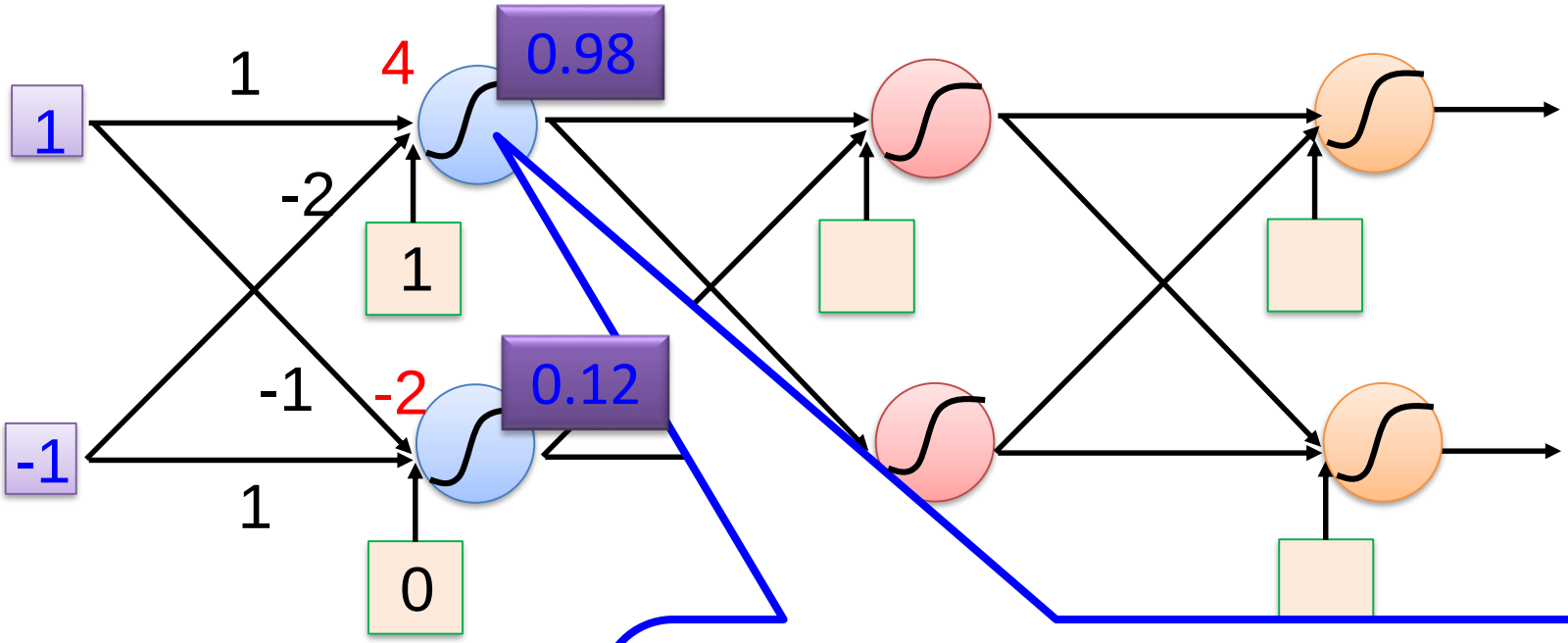
Function of Neural Network

- If the number of hidden layer > 1, we call “deep” neural network.



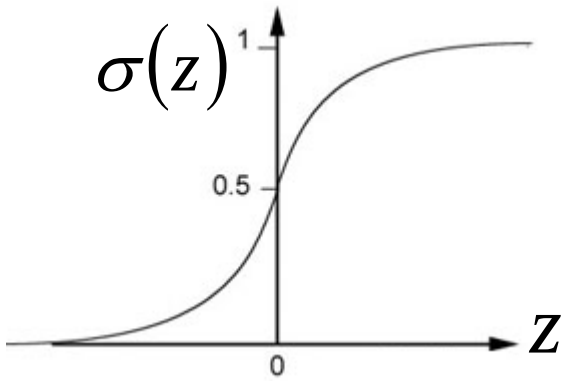
$$y = f(x)$$
$$= \sigma\left(W^L \ldots \sigma\left(W^2 \sigma\left(W^1 x + b^1\right) + b^1\right) \ldots + b^L\right)$$

Example of Neural Network (1/3)

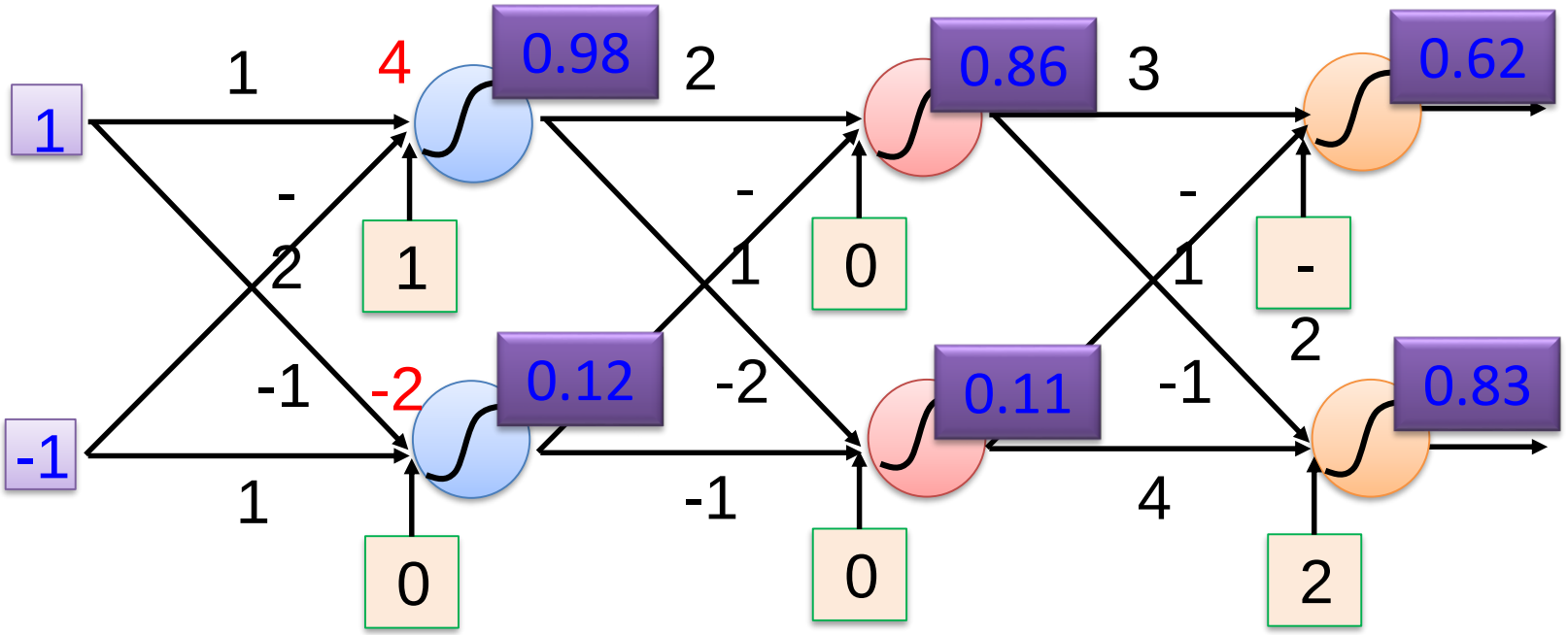


Sigmoid Function

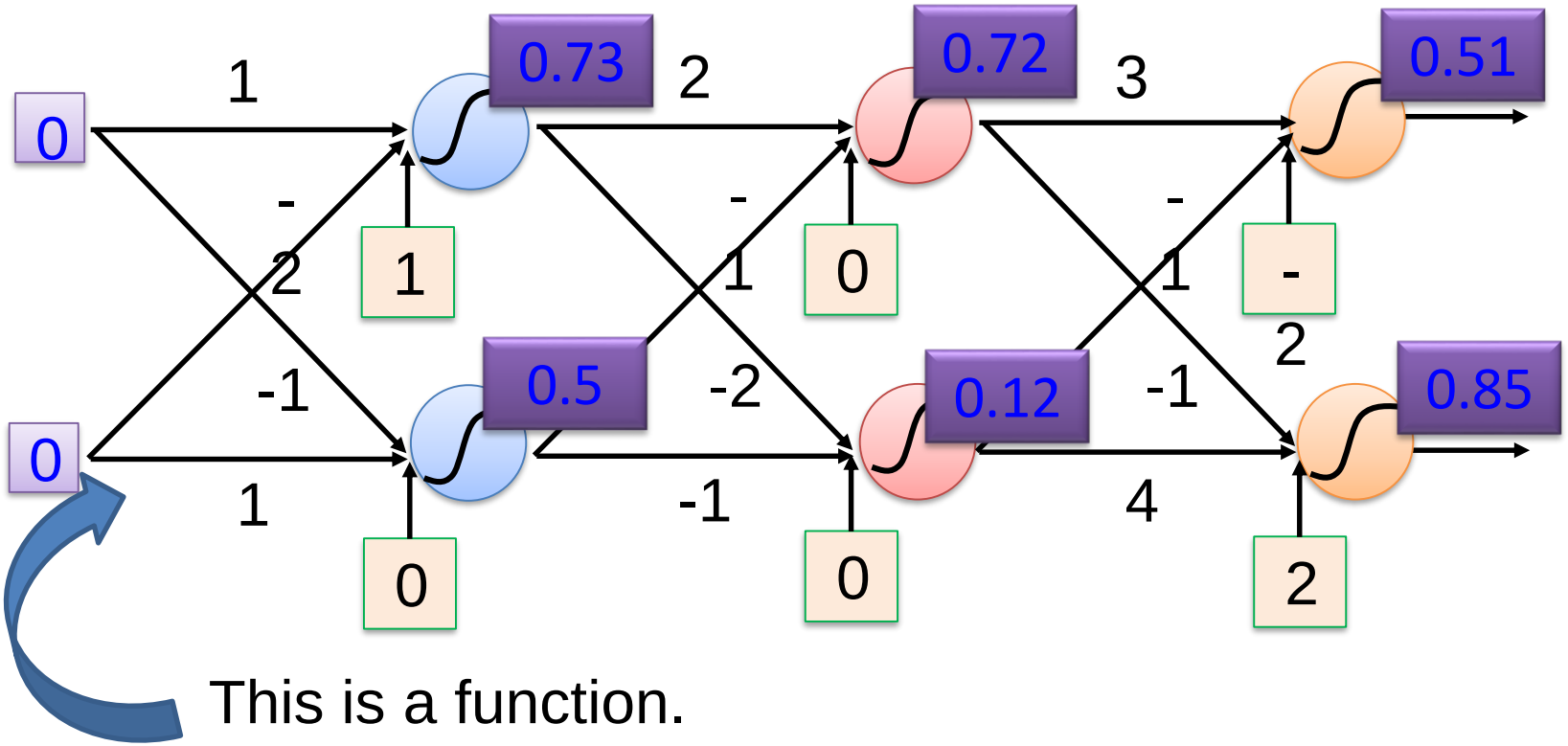
$$\sigma(z) = \frac{1}{1+e^{-z}}$$



Example of Neural Network (2/3)



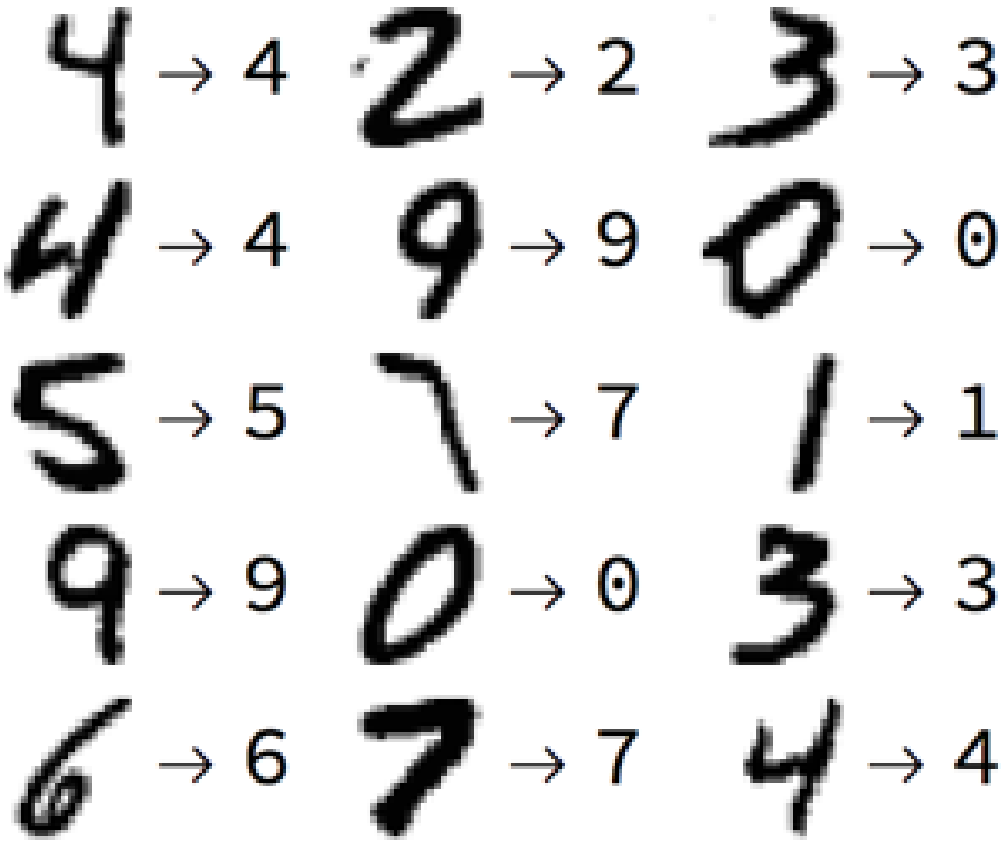
Example of Neural Network (3/3)

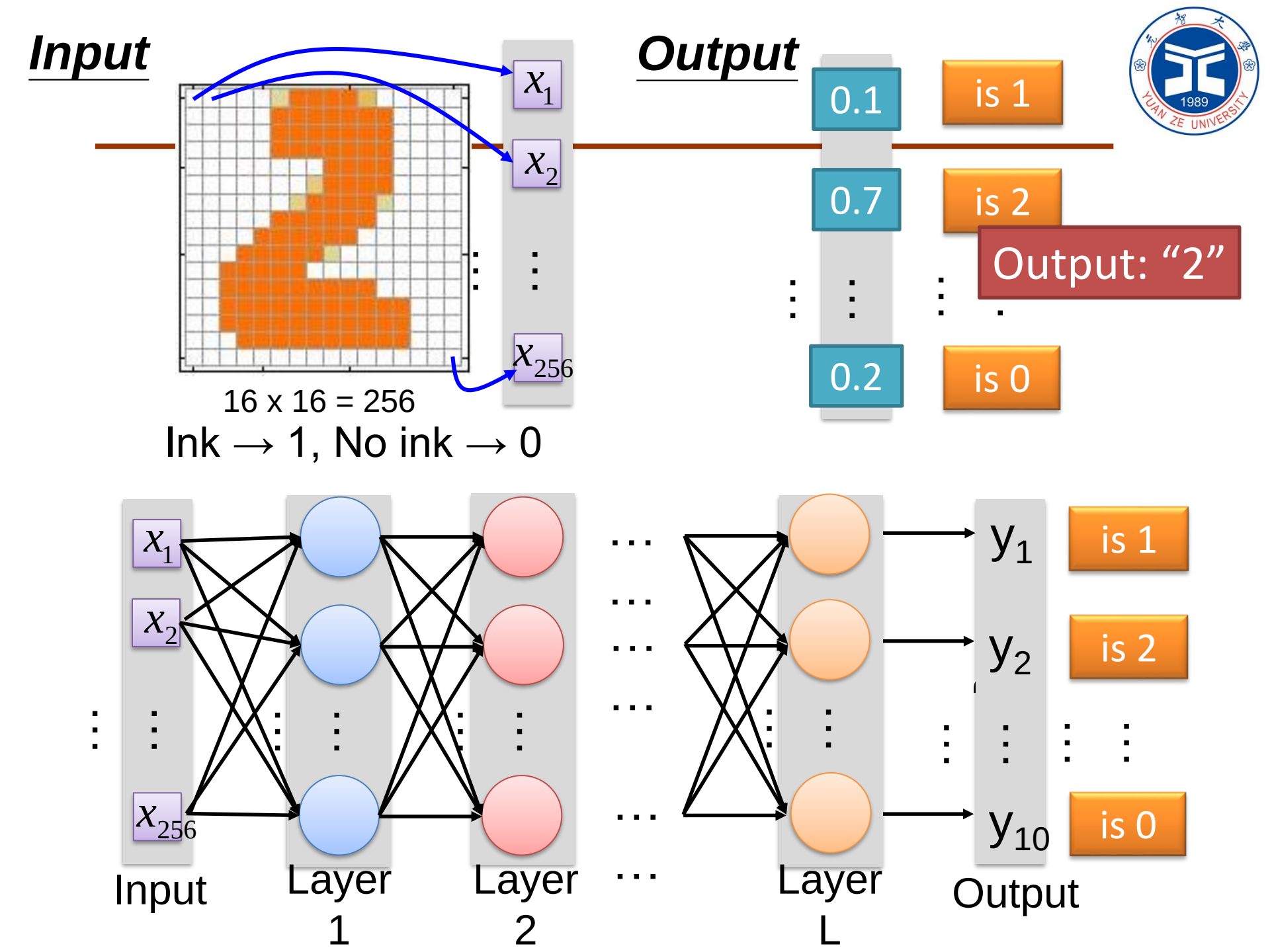


Input vector, output vector

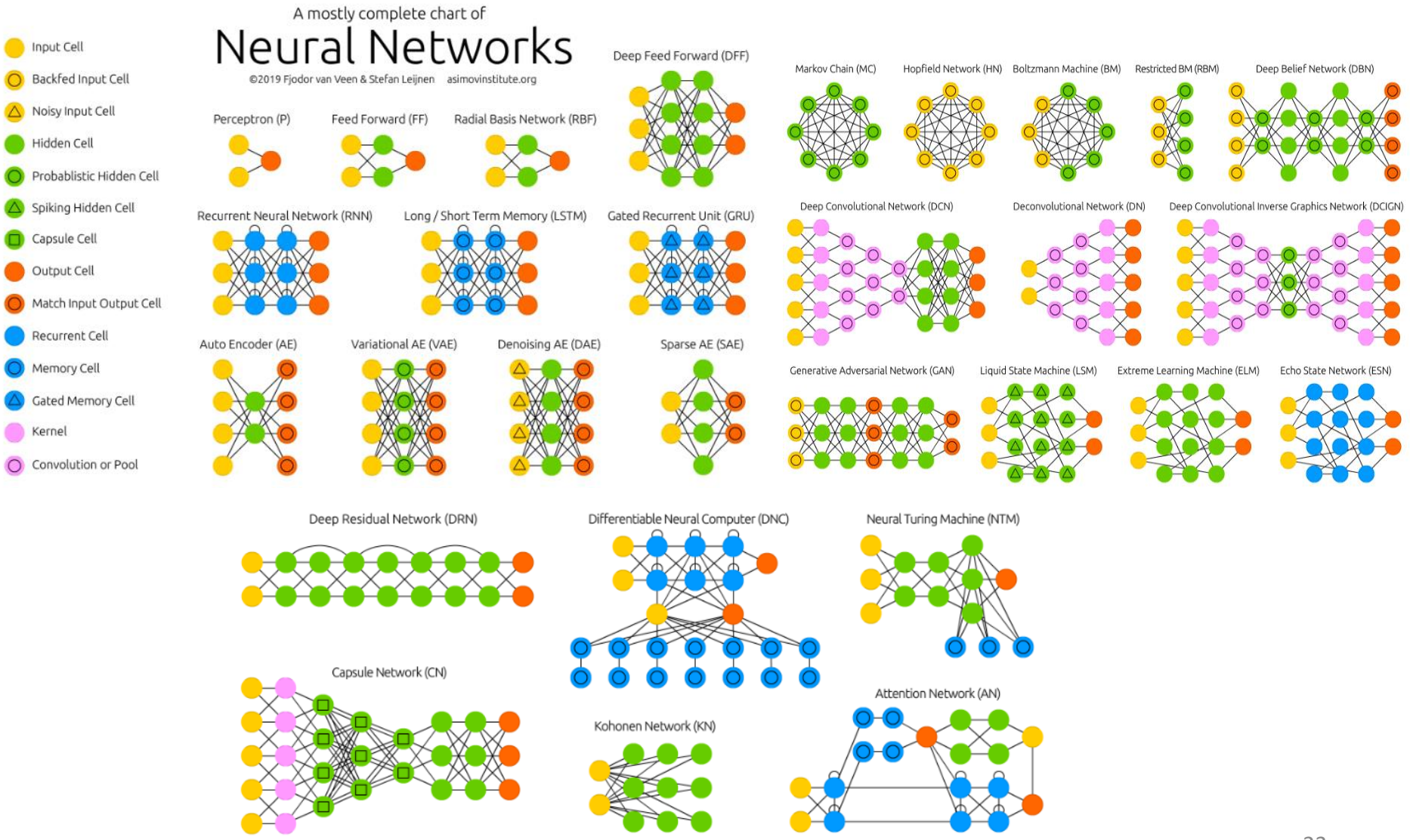
$$f\left(\begin{bmatrix}1\\-1\end{bmatrix}\right)=\begin{bmatrix}0.62\\0.83\end{bmatrix} \quad f\left(\begin{bmatrix}0\\0\end{bmatrix}\right)=\begin{bmatrix}0.51\\0.85\end{bmatrix}$$

Example: Handwriting Recognition




Input
Output
x_1
x_2
x_{256}
16 x 16 = 256
Ink → 1, No ink → 0
0.1
is 1
0.7
is 2
Output: "2"
0.2
is 0
x_1
x_2
x_{256}
y_1
y_2
y_{10}
is 1
is 2
is 0
Input
Layer 1
Layer 2
Layer L
Output

Different Kind of Neural Network



Interesting Project Review (with Python)

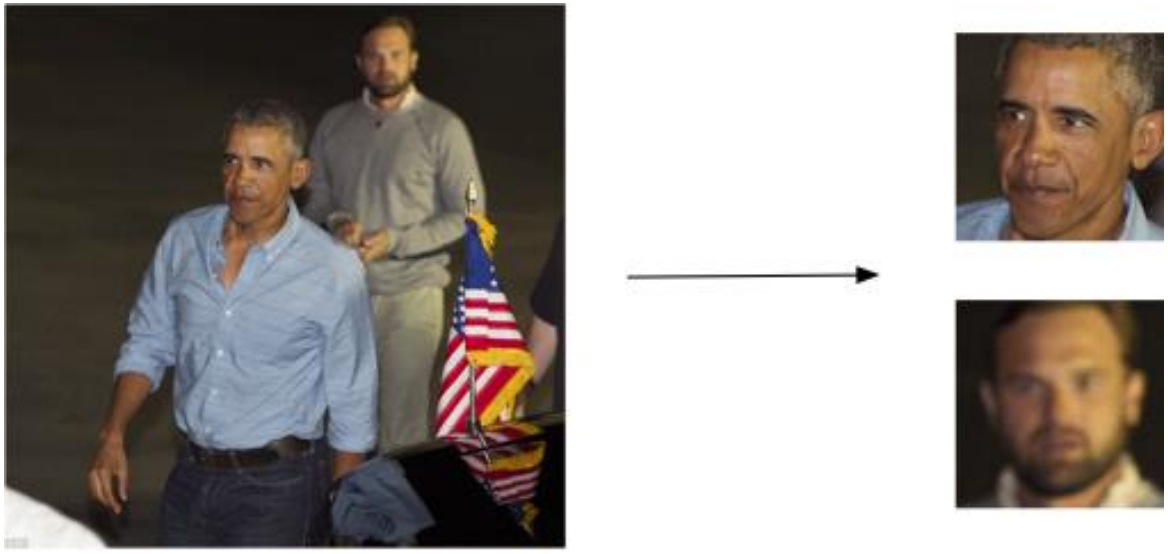
2017-2018

Face Recognition

- Built using dlib's state-of-the-art face recognition built with deep learning.
- The model has an accuracy of 99.38% on the Labeled Faces in the Wild benchmark.
- Features:
  - **Find faces in pictures**
  - **Find and manipulate facial features in pictures**
  - **Identify faces in pictures**
- https://github.com/ageitgey/face_recognition?utm_source=mybridge&utm_medium=blog&utm_campaign=read_more

Find Faces in Pictures

```
import face_recognition
image = face_recognition.load_image_file("your_file.jpg")
face_locations = face_recognition.face_locations(image)
```



Input

Output

Find and manipulate facial features in pictures

- Get the locations and outlines of each person's eyes, nose, mouth and chin.

Input

Output

```
import face_recognition
image = face_recognition.load_image_file("your_file.jpg")
face_landmarks_list = face_recognition.face_landmarks(image)
```

Applying Digital Make-up

Input

Output

Identify Faces in Pictures

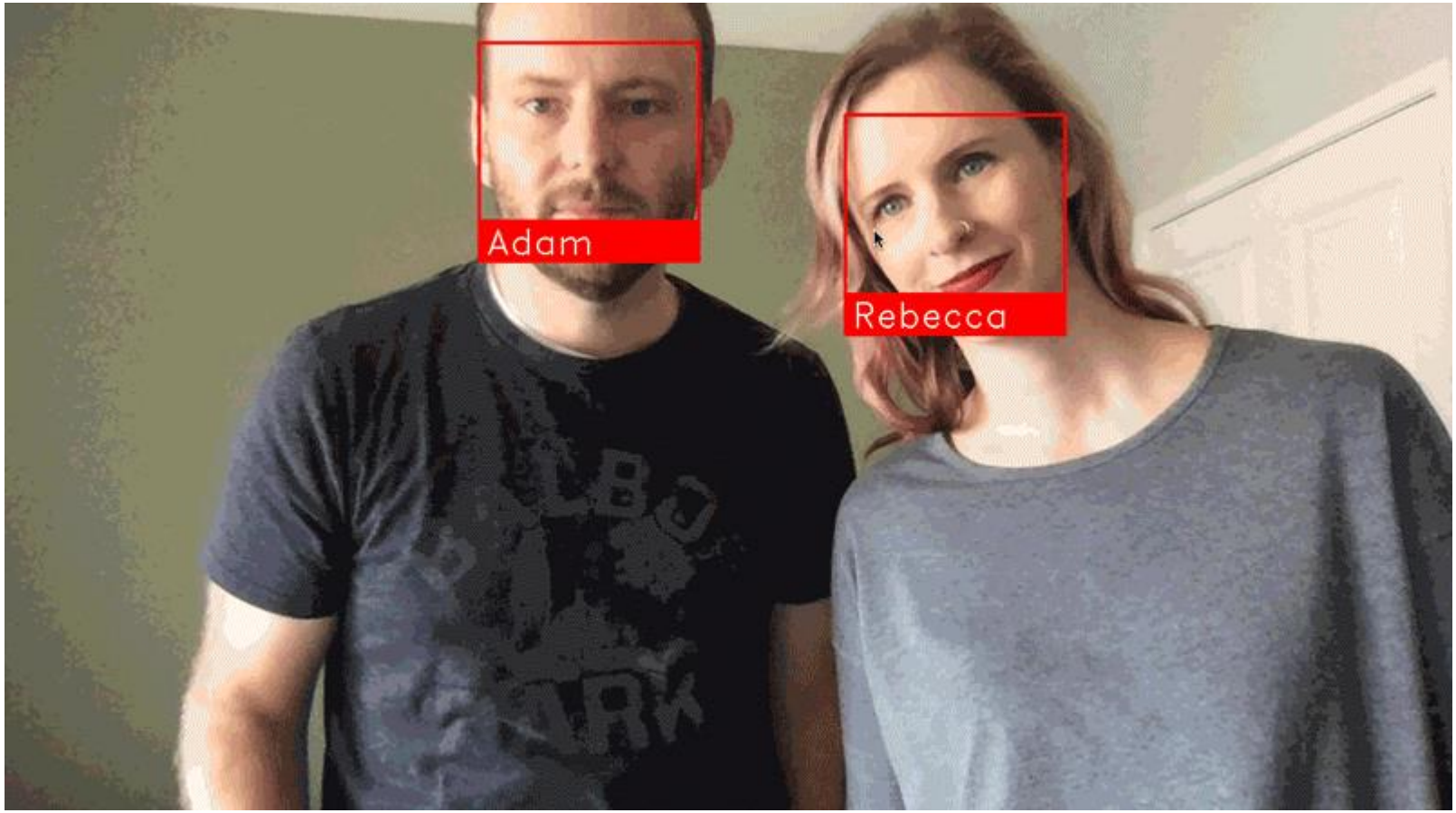
Input

Picture contains
"Joe Biden"

Output

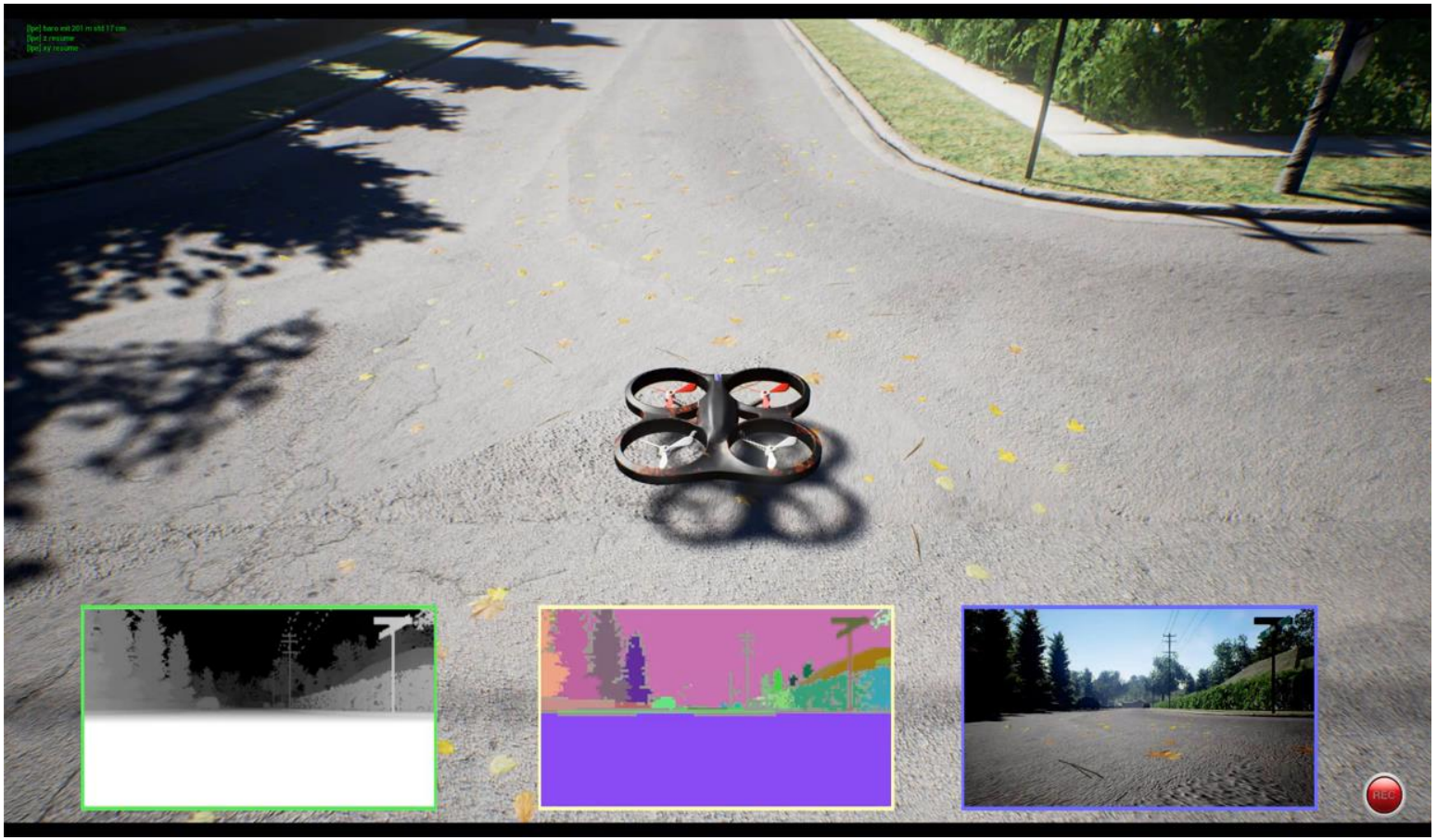
Real-time Face Recognition

- You can even use this library with other Python libraries to do real-time face recognition.



AirSim

- Open source simulator based on Unreal Engine for autonomous vehicles from Microsoft AI & Research



https://github.com/Microsoft/AirSim

Voice Conversion with Non-Parallel Data

- Deep neural networks for voice conversion (voice style transfer) in Tensorflow
- What if you could imitate a famous celebrity's voice or sing like a famous singer?

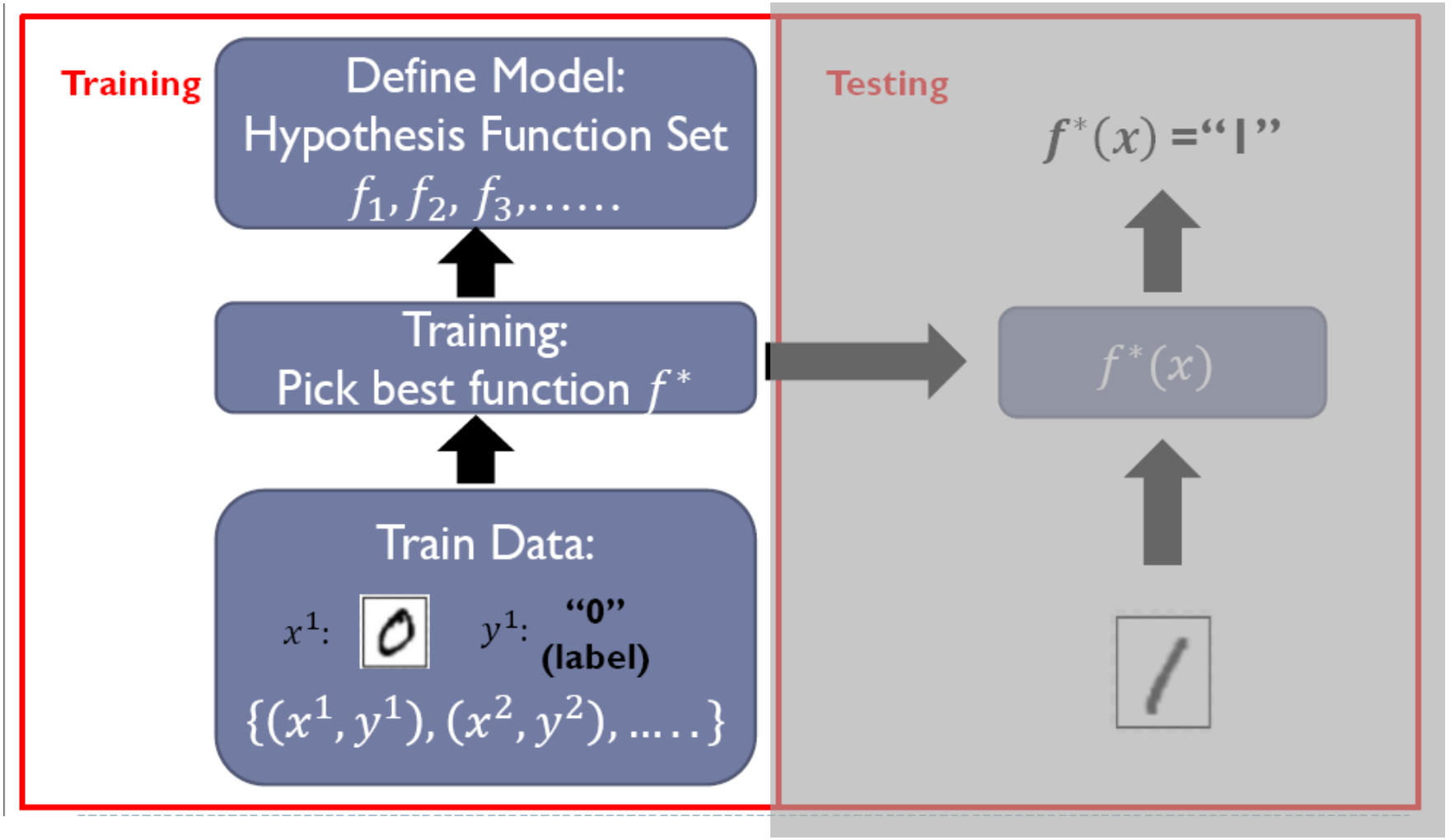
Sample Link

https://github.com/andabi/deep-voice-conversion

Outline

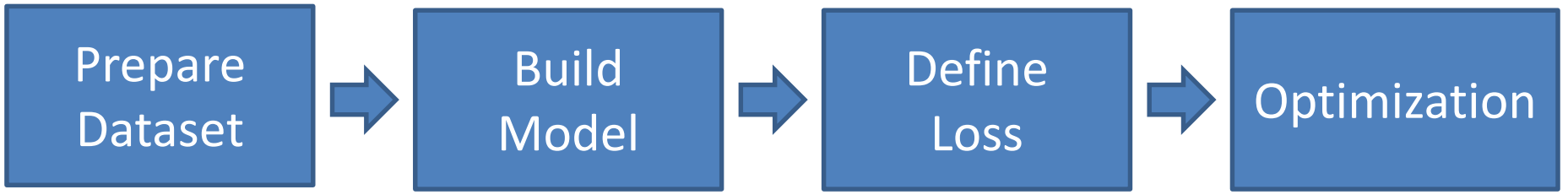
- Training Model
  - Prepare dataset
  - Build model
  - Define loss
    - Mean Square
    - Cross-entropy
  - Optimization
    - Gradient decent
    - Increase the performance of optimization
      - Tips 1:Tune Learning Rate
      - Tips2: Stochastic Gradient Descent
      - Tips3: Feature Scaling
    - How to update many parameters
- Testing Model
- DNN Discussion

Learning


Training
Define Model:
Hypothesis Function Set
$f_1, f_2, f_3, \cdots\cdots$
Training:
Pick best function f^*
Train Data:
x^1:
y^1:
"0"
(label)
$\{(x^1, y^1), (x^2, y^2), \ldots\ldots\}$
Testing
$f^*(x)$ ="1"
$f^*(x)$

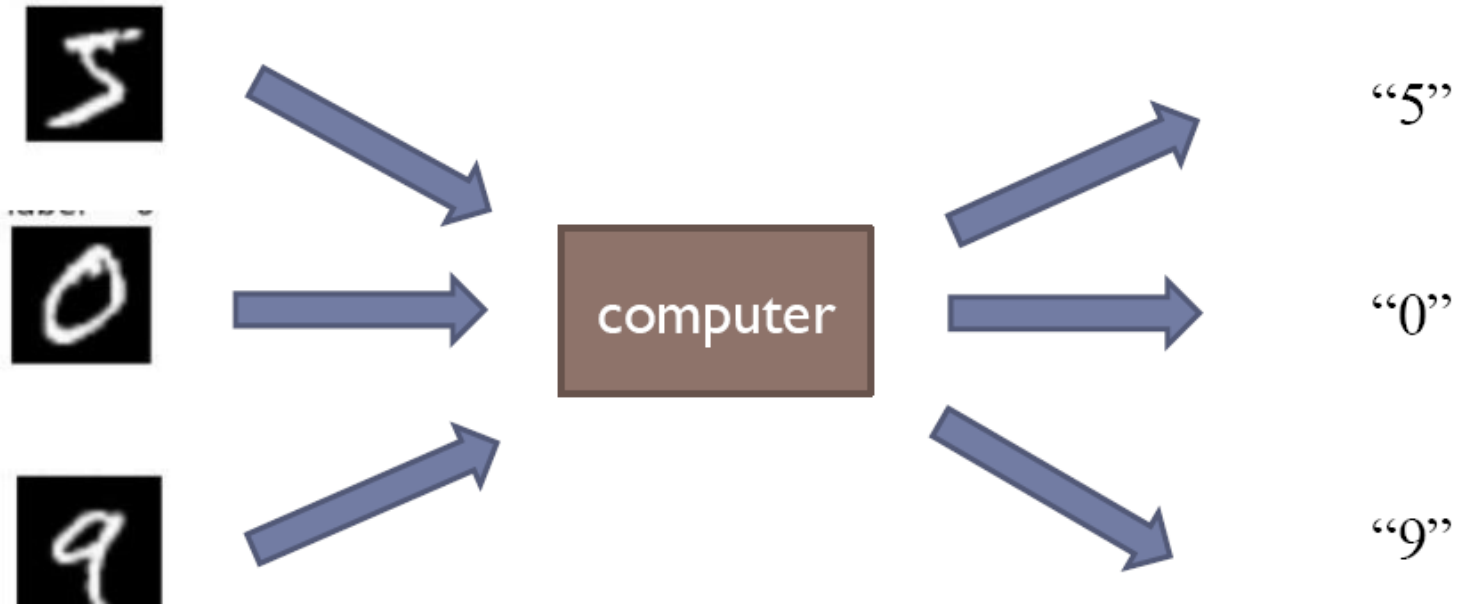
The Steps of Training Model

- No matter which NN, the steps are the same.
  - Prepare dataset
  - Build model
  - Define loss
  - Optimization



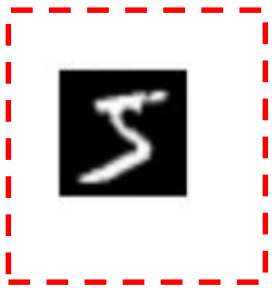
Example

- Assume we want to build a handwritten system to recognize image from 0 to 9



1. Preparing Dataset

1. Preparing Dataset

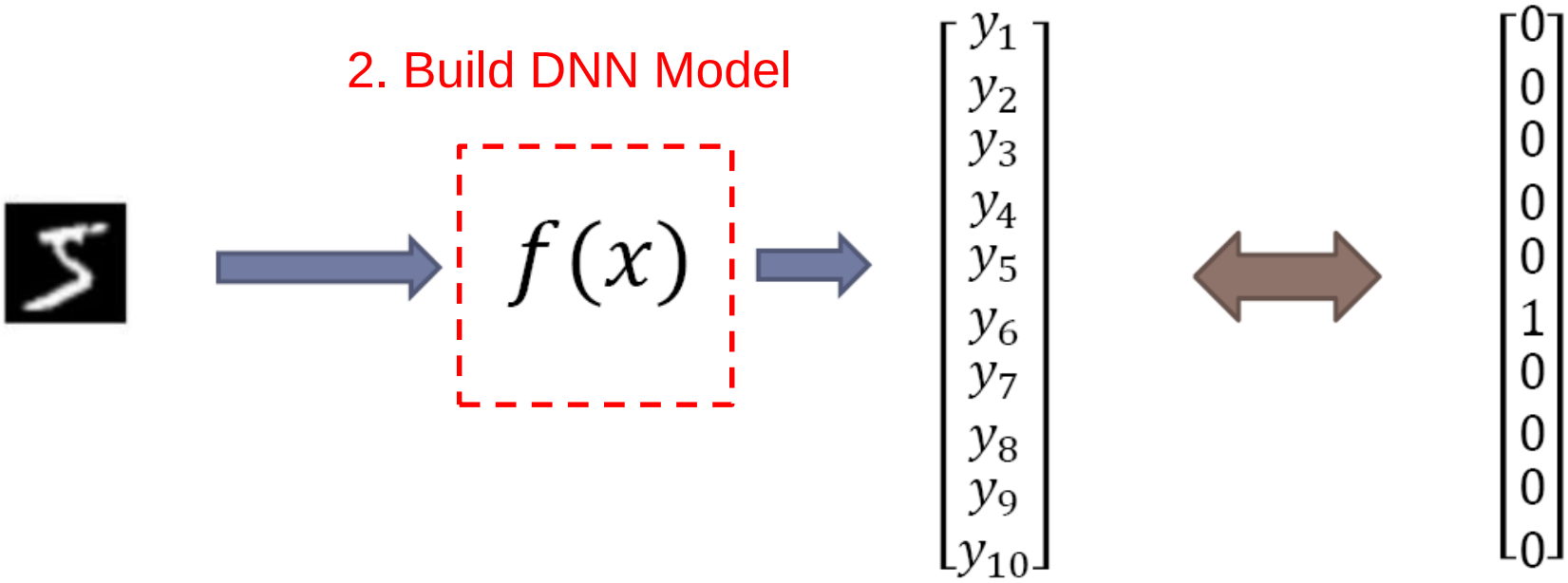


$f(x)$

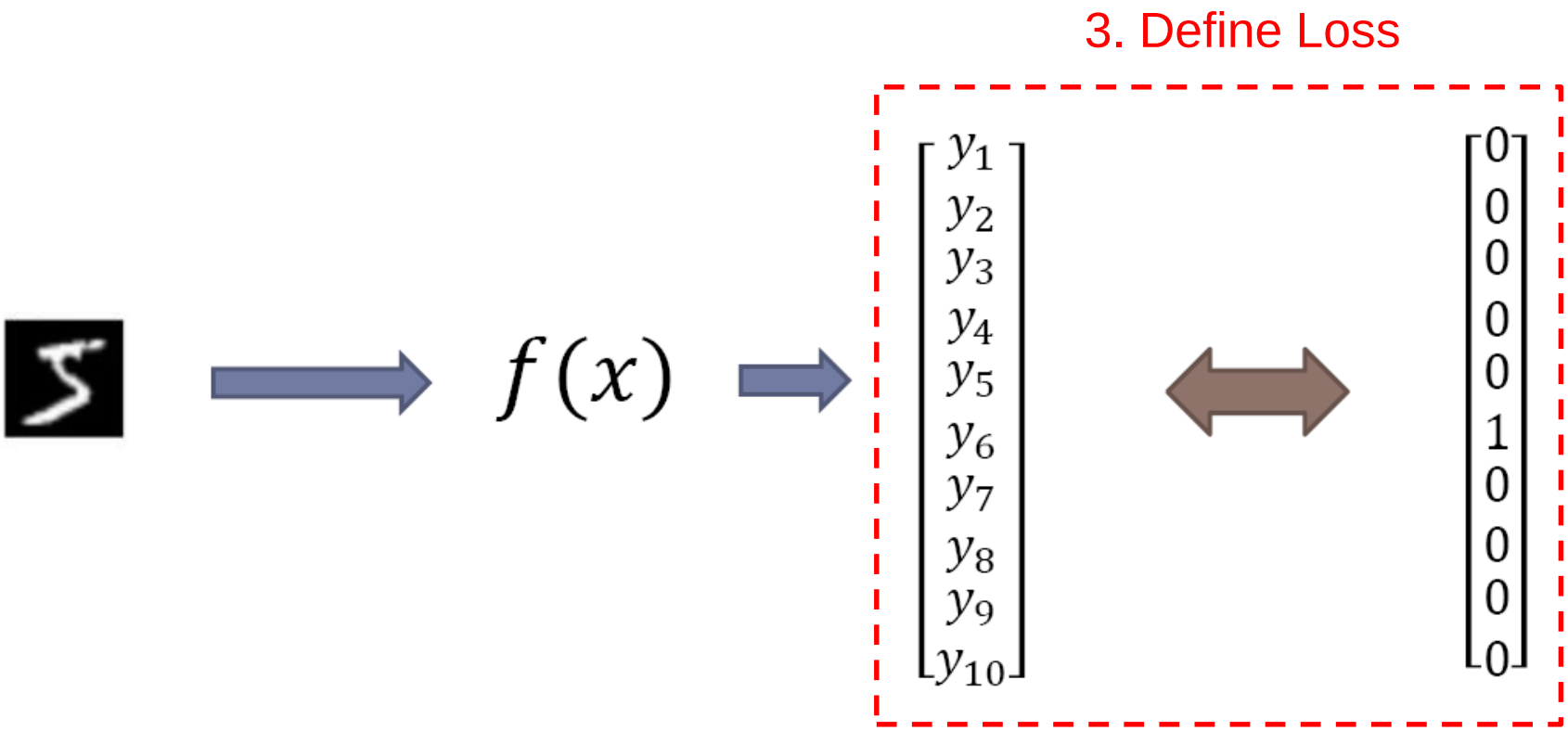
$$\begin{bmatrix} y_1 \\ y_2 \\ y_3 \\ y_4 \\ y_5 \\ y_6 \\ y_7 \\ y_8 \\ y_9 \\ y_{10} \end{bmatrix} \Leftrightarrow \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 1 \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix}$$

We want this two as close as possible.

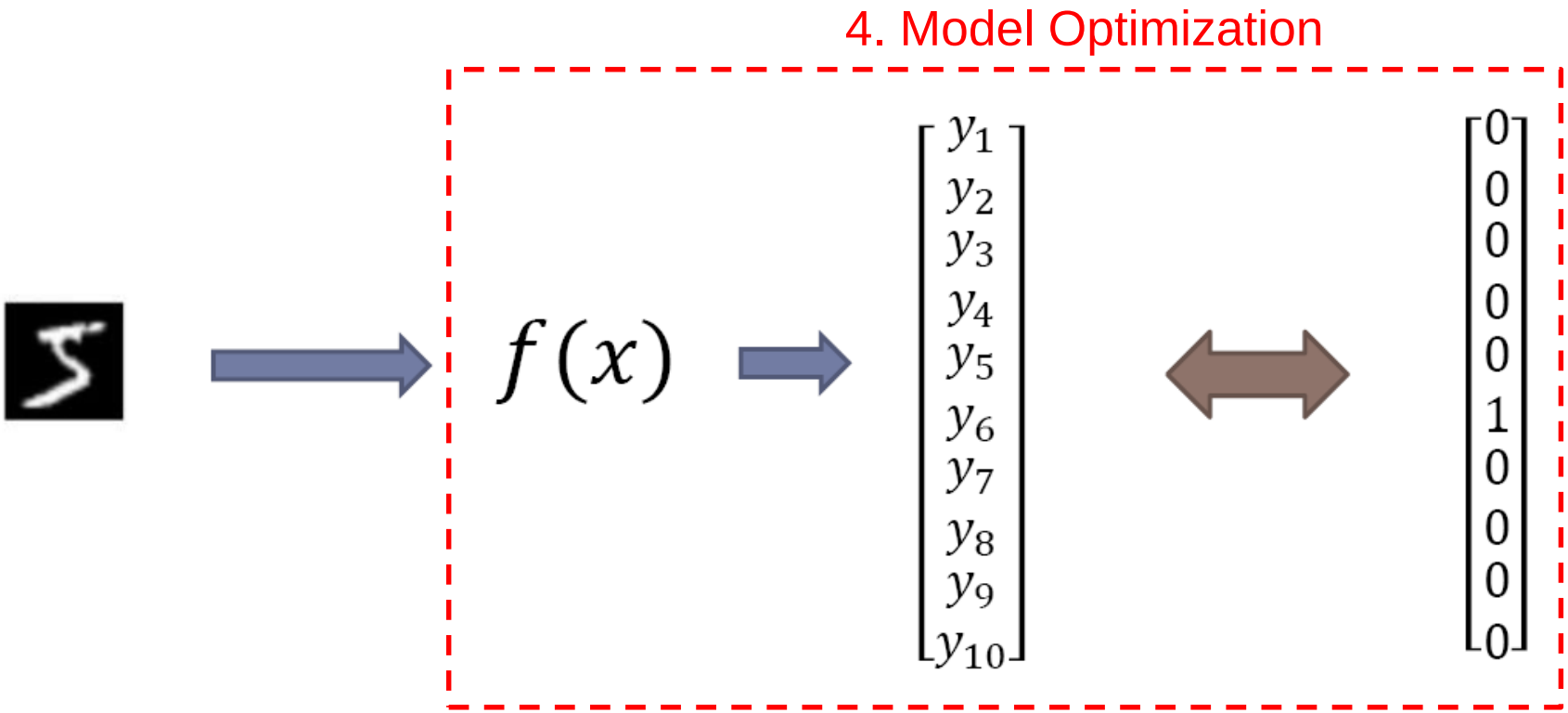
2. Build DNN Model



3. Define Loss



4. Model Optimization



Result

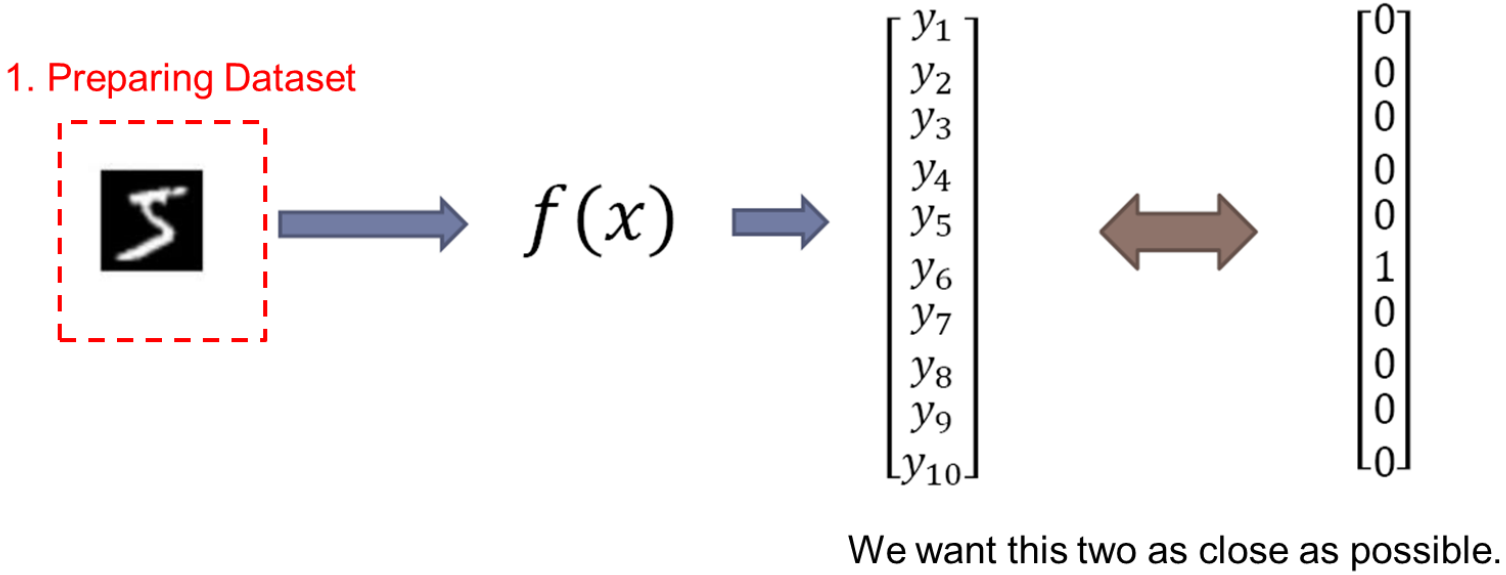
$f^*(x)$

$$\begin{bmatrix} 0 \\ 0.2 \\ 0 \\ 0.1 \\ 0 \\ 0.8 \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix}$$

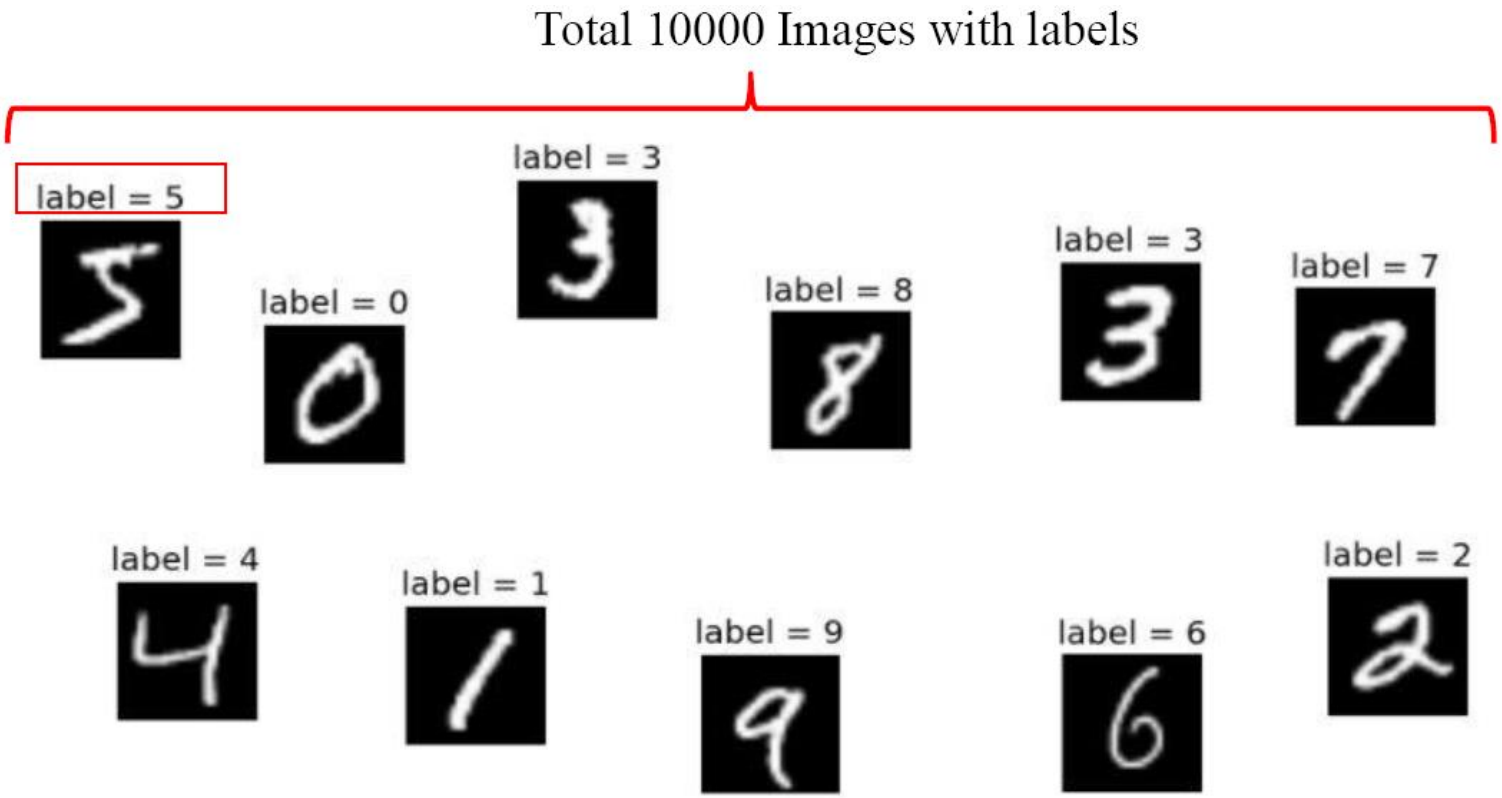
We get it is '5'

Steps

- 1. Prepare Dataset
- 2. Build Model
- 3. Define Loss
- 4. Model Optimization

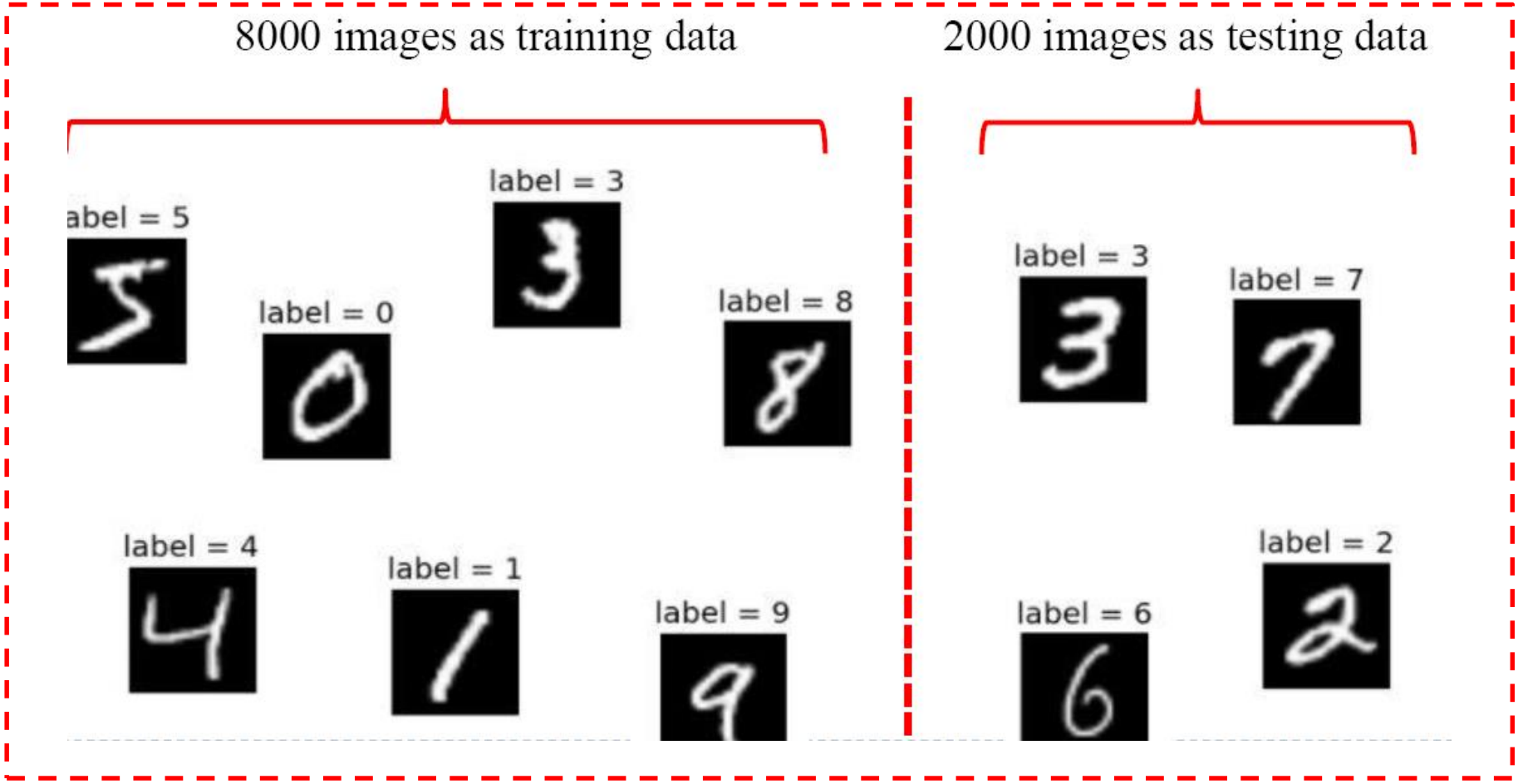


Preparing Dataset



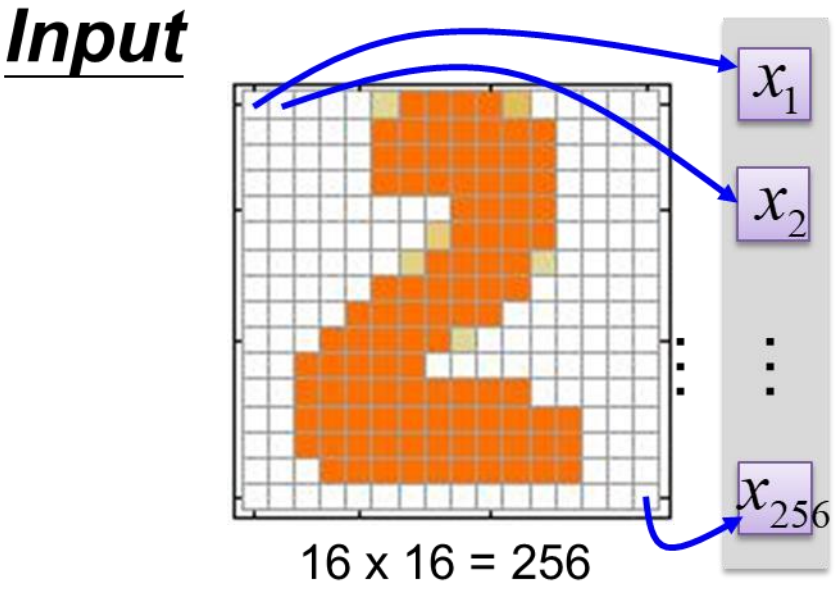
Split the data

1. Prepare Dataset

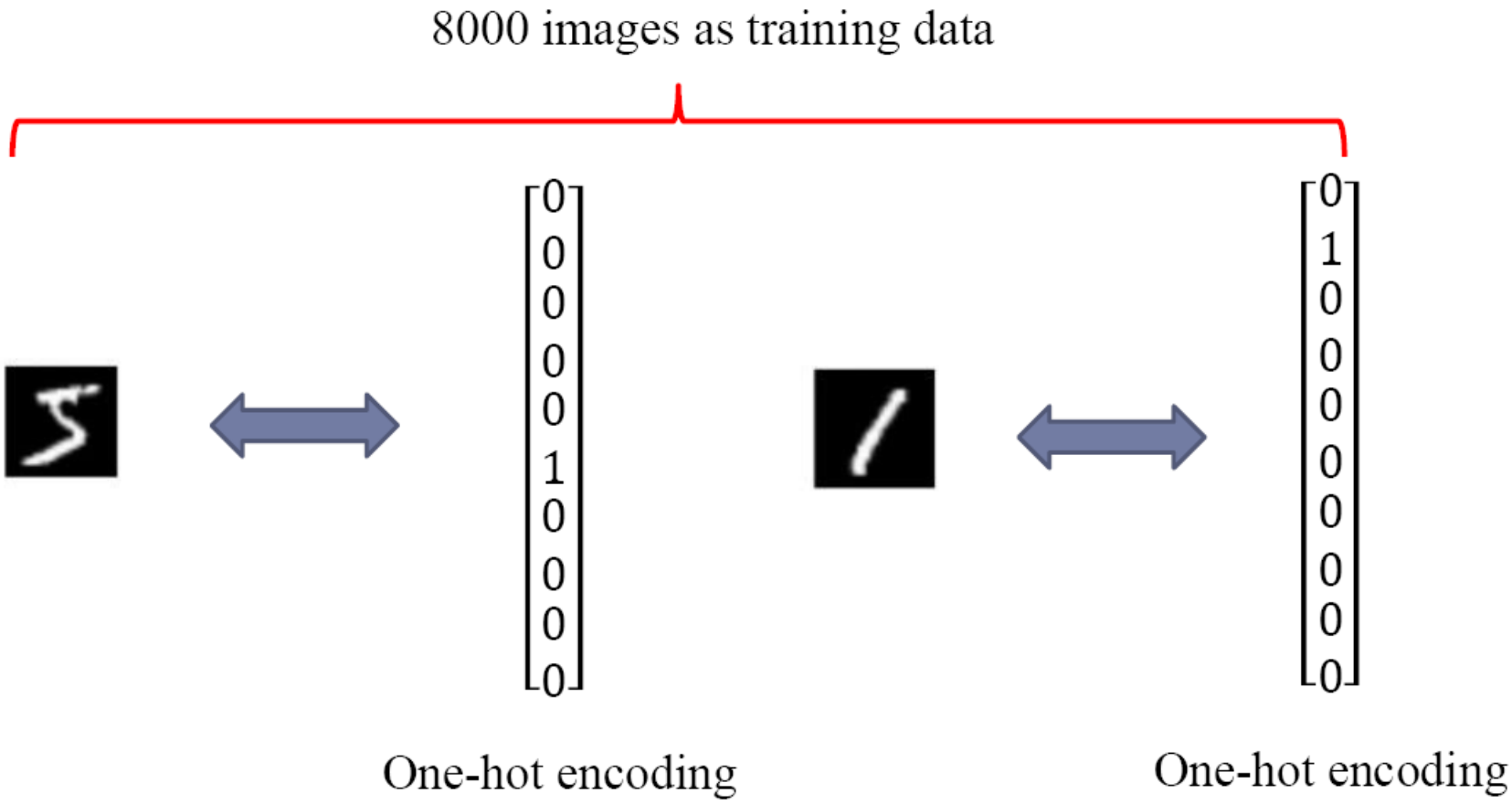


Transfer Input to Vector

- Ink ->1
- No Ink ->0



Encode Labels



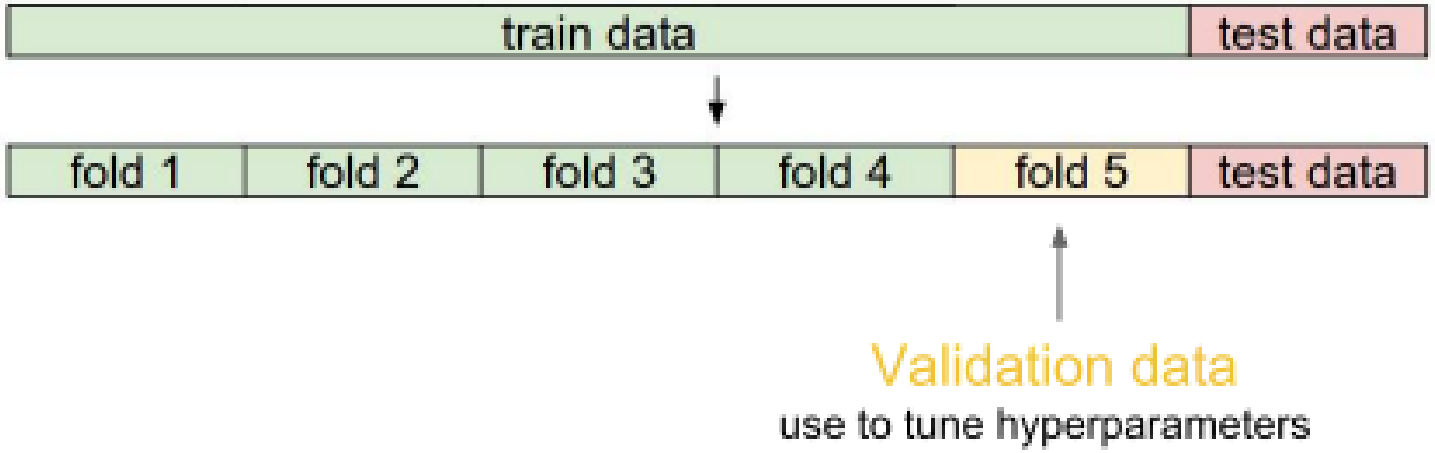
Training set, Validation set, Test set (1/2)

- Try out what hyper parameters work best on test set.

Very bad idea!!
The test set is the proxy for the generalization performance!
Use only “VERY SPARINGLY,” at the end.

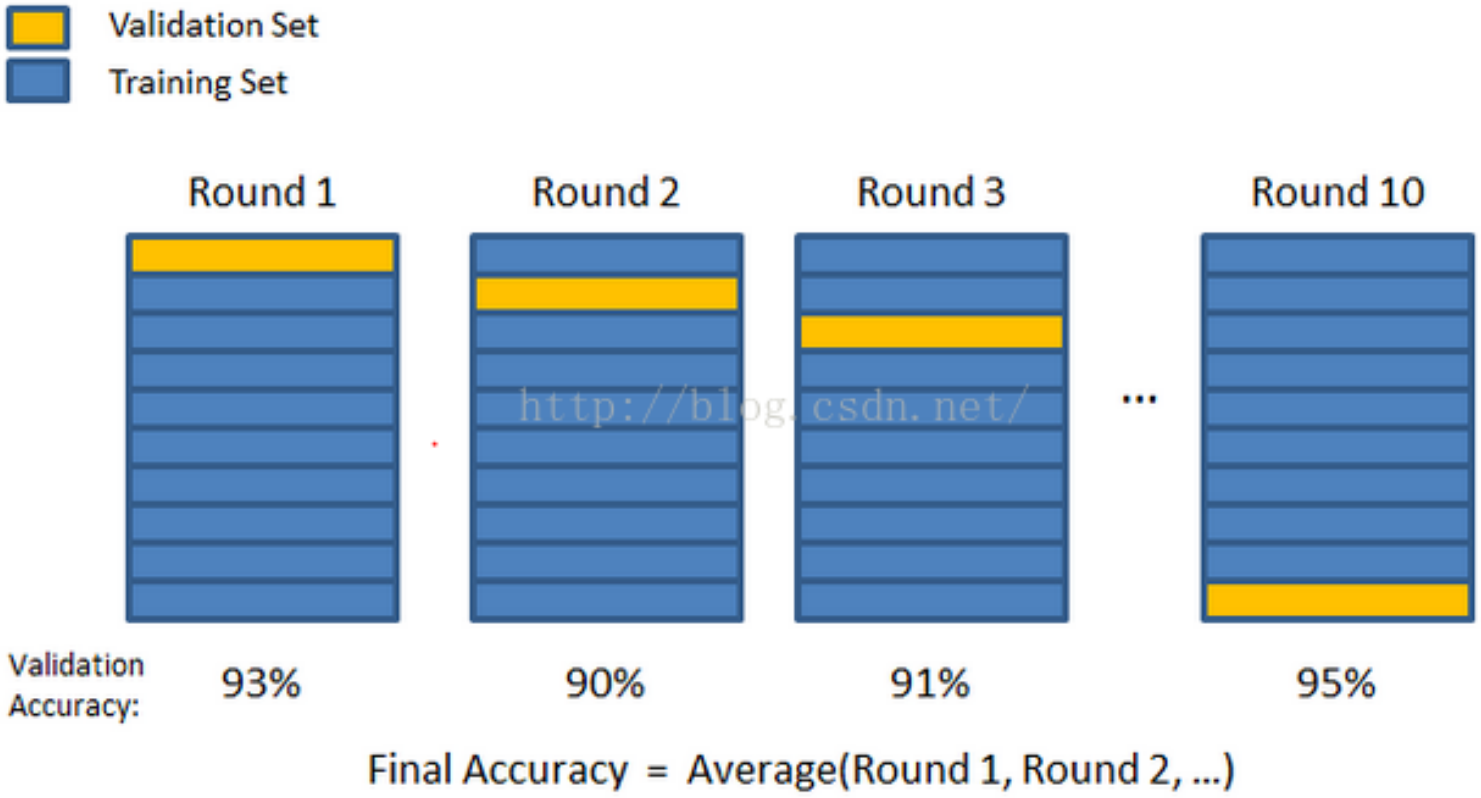
Training set, Validation set, Test set

- Split part of training data as validation data.
- Using training data to train model.
- Using validation data to validate model and modify model.



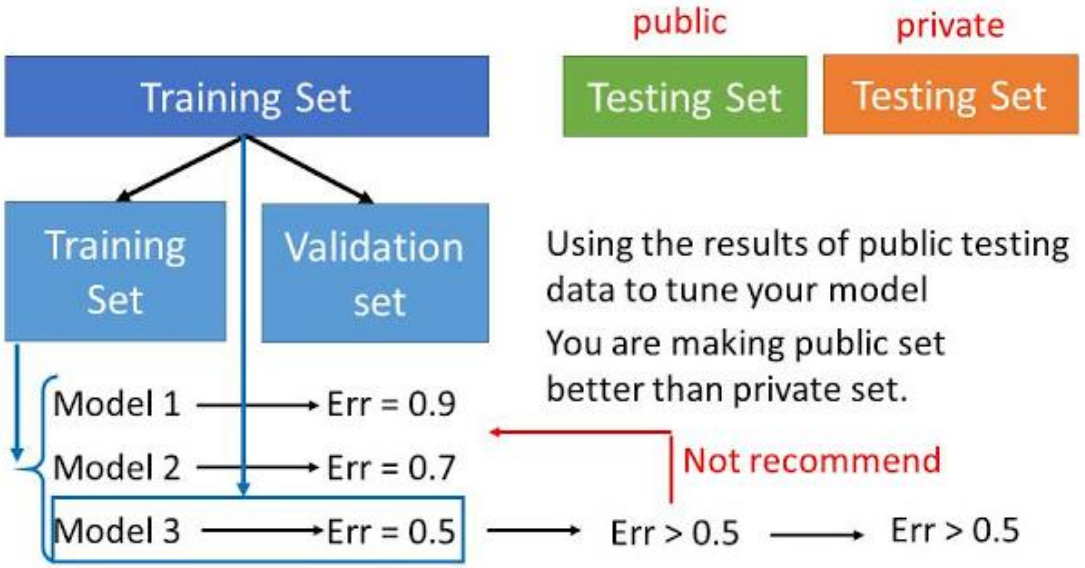
Cross Validation

- Split training data and validation set => Cross Validation



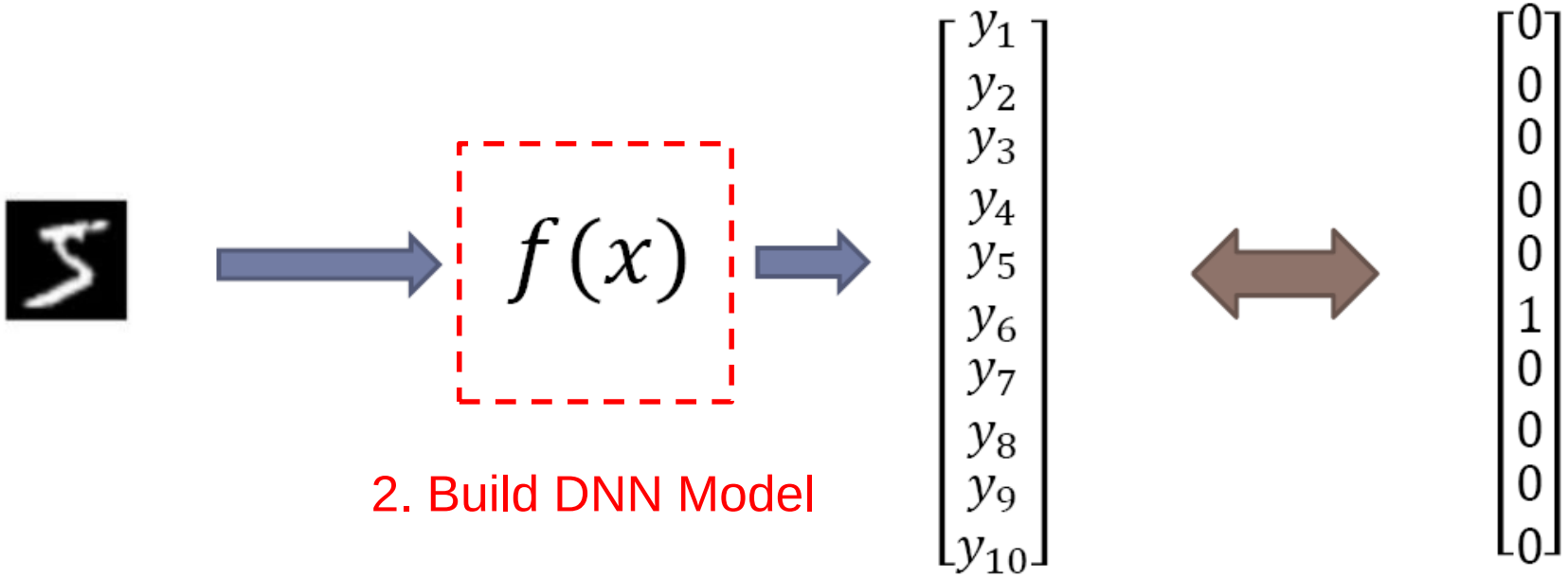
Training set, Validation set, Test set (2/2)

- After modifying model many times, best model is built.
- Using test data to test the model.
- There are public test data and private test data.
- The difference of test data and validation data is we don't use test data to modify model. Test data only let us know the model performance under real scenario.

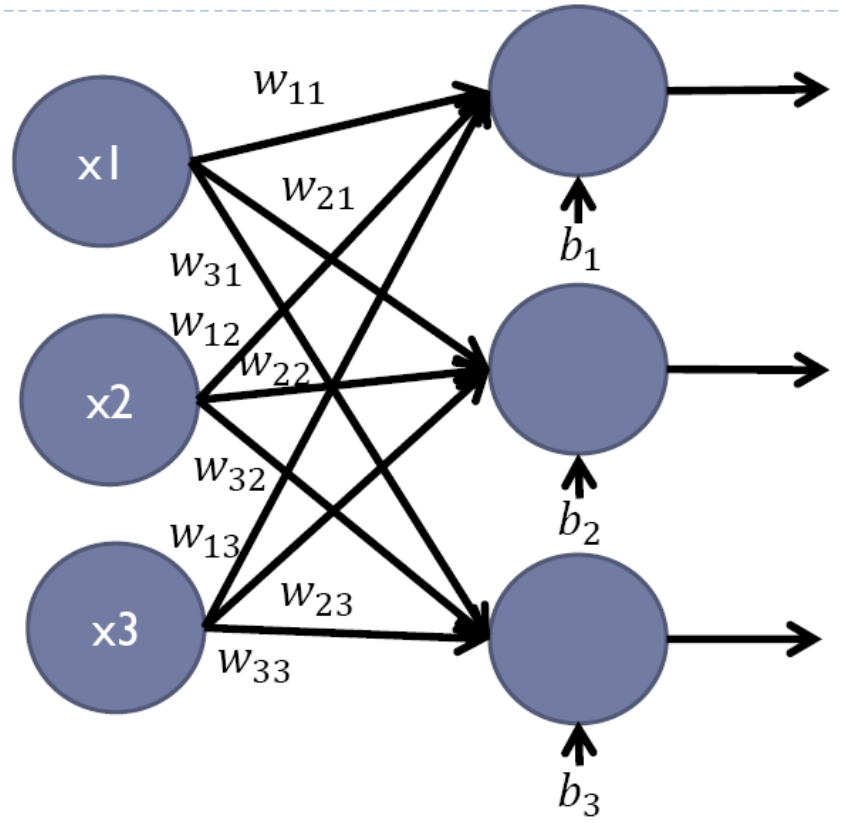


Steps

- 1. Prepare Dataset
- 2. Build DNN Model
- 3. Define Loss
- 4. Optimization



Model Overview

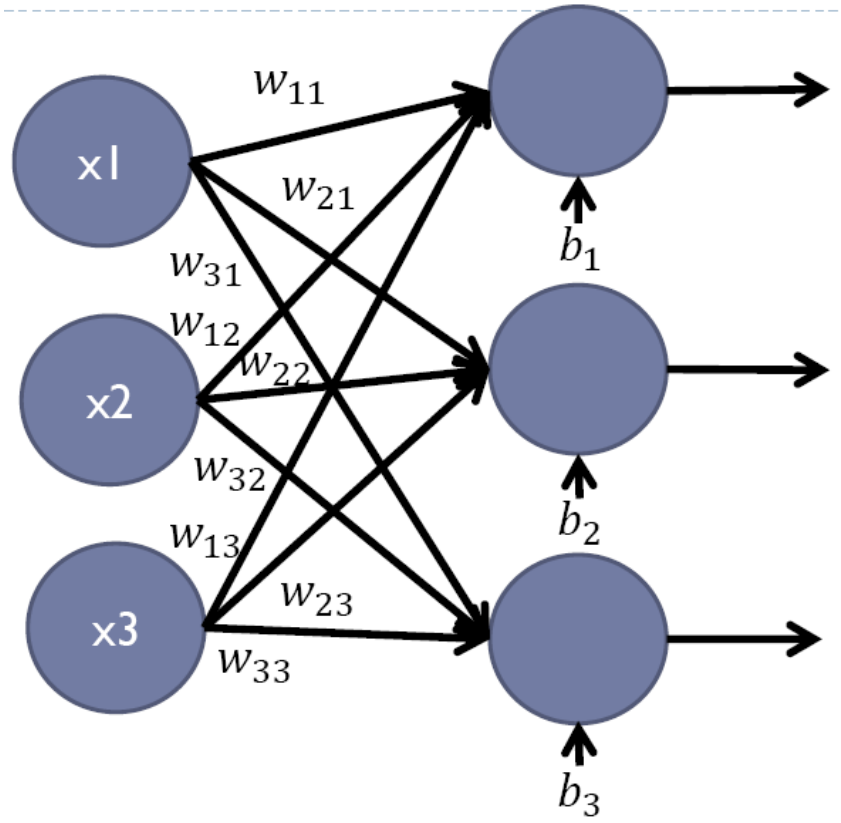


$$\sigma(x1 * w_{11} + x2 * w_{12} + x3 * w_{13} + b_1)$$

$$\sigma(x1 * w_{21} + x2 * w_{22} + x3 * w_{23} + b_2)$$

$$\sigma(x1 * w_{31} + x2 * w_{32} + x3 * w_{33} + b_3)$$

Model Overview

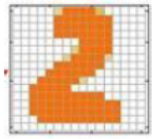


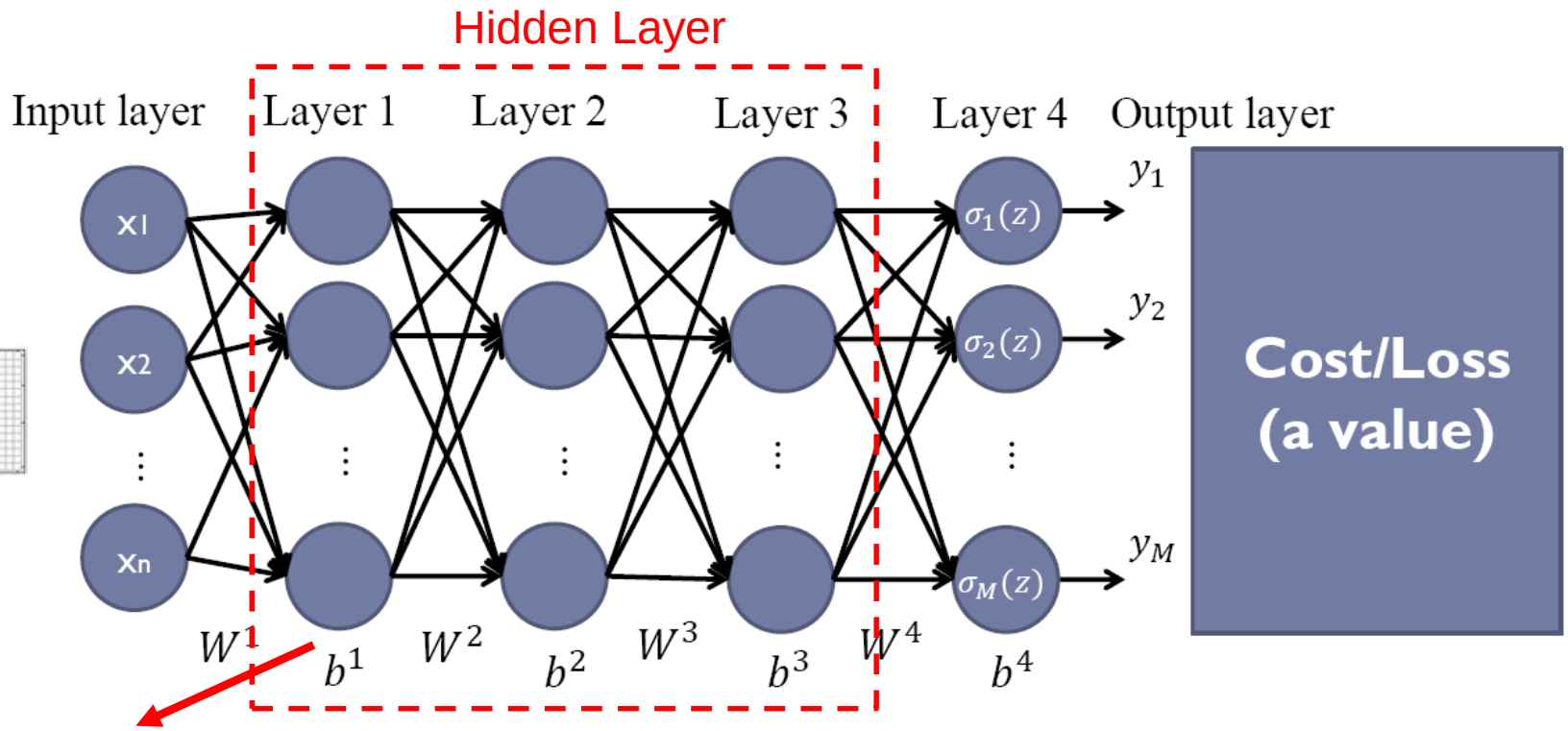
$$\sigma(W * X + b)$$

$$X = \begin{bmatrix} x1 \\ x2 \\ x3 \end{bmatrix} \qquad W = \begin{bmatrix} w_{11} & w_{12} & w_{13} \\ w_{21} & w_{22} & w_{23} \\ w_{31} & w_{32} & w_{33} \end{bmatrix}$$

$$b = \begin{bmatrix} b1 \\ b2 \\ b3 \end{bmatrix}$$

DNN Model Overview



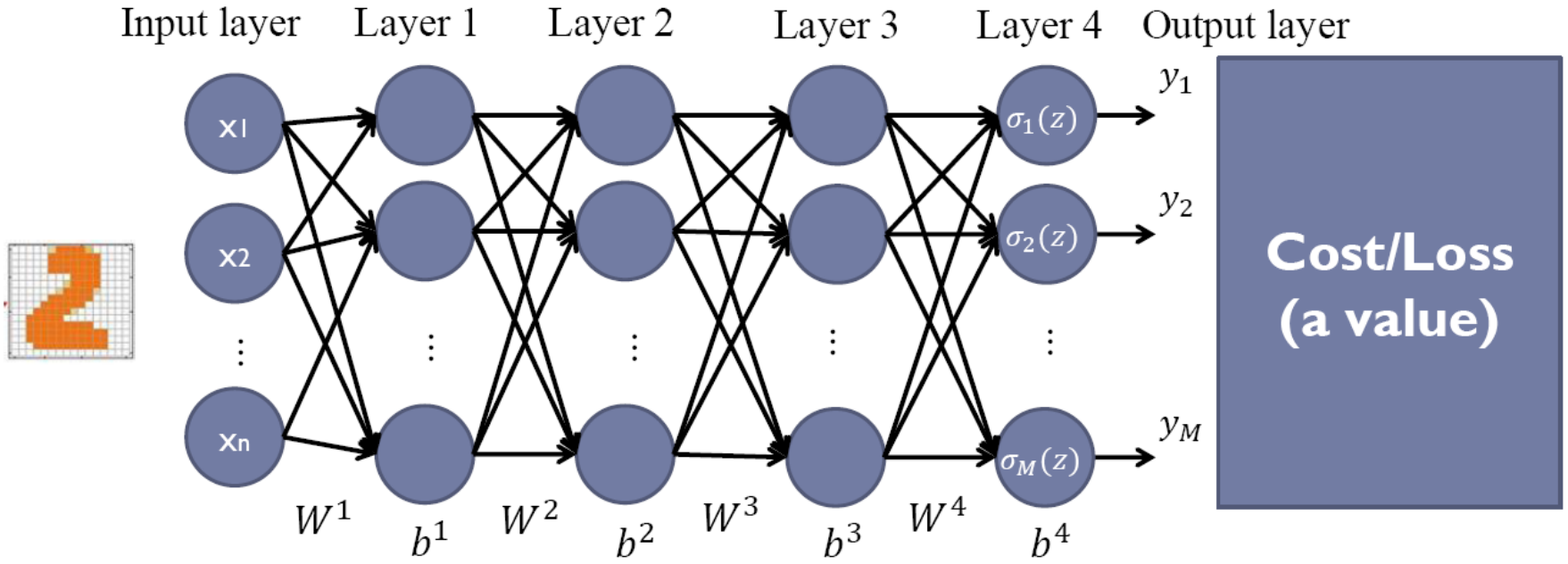


DNN also called "Multilayer Perceptron"

$$y = f(x) = \sigma(W^L \ldots \sigma(W^2\sigma(W^1X + b^1) + b^2) \ldots + b^L)$$

Activation function Weight (variable) Bias (variable)

DNN Model Overview



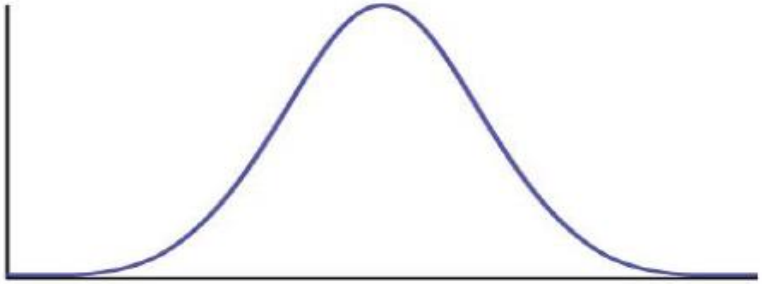
$$y = f(x) = \sigma(W^L \ldots \sigma(W^2 \sigma(W^1 X + b^1) + b^2) \ldots + b^L)$$

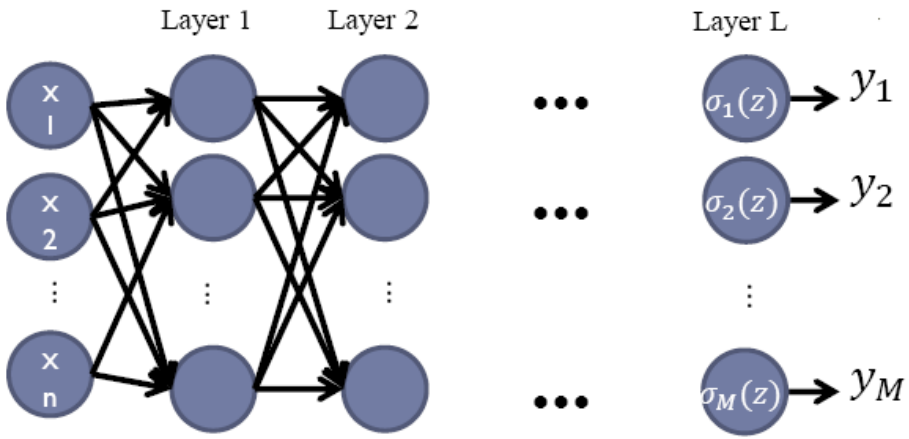
We want to find the best parameter set :

$$\theta = \{W^1, b^1, W^2, b^2, \ldots, W^L, b^L\}$$

Parameter Initialization

- The most simple way are
  - Set all of them to zero (sometimes will fail)
  - Use random normal distribution with zero mean and small standard deviation
- In complex model, it is very important to find better way to initialize the parameters.

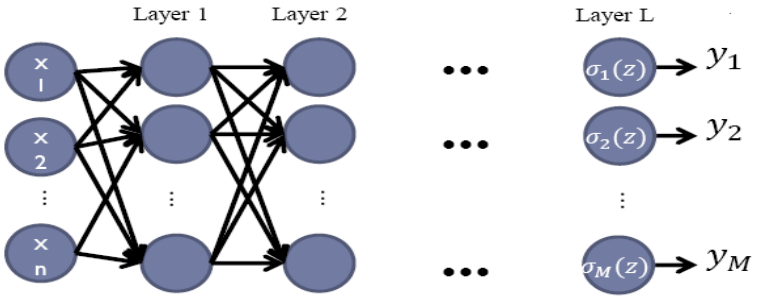




$$\theta = \{W^1, b^1, W^2, b^2, \ldots, W^L, b^L\}$$

Activation Function

- Give NN nonlinear property
- Can be regarded as "ON(1) or OFF(0)"



$$\theta = \{W^1, b^1, W^2, b^2, \ldots, W^L, b^L\}$$

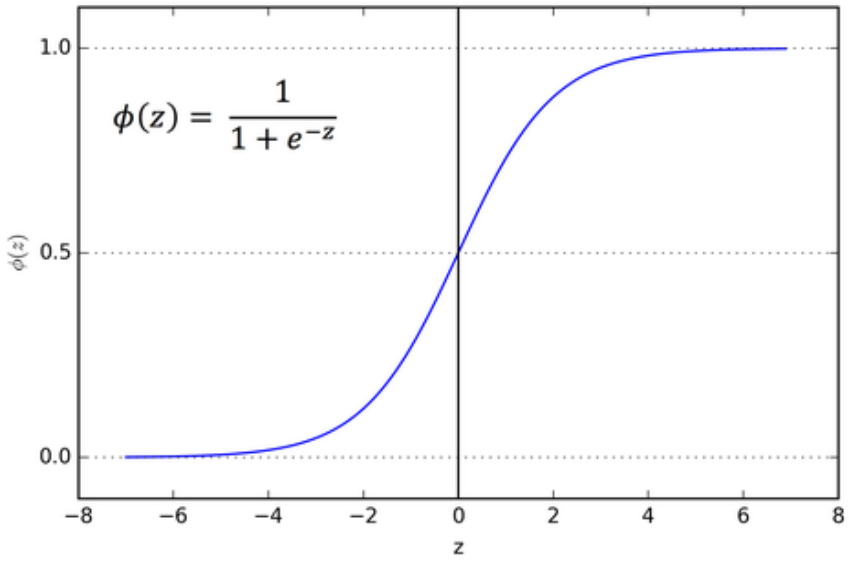
$$y = f(x) = \sigma(W^L \ldots \sigma(W^2\sigma(W^1X + b^1) + b^2) \ldots + b^L)$$

Activation function Weight (variable) Bias (variable)

- There are many kinds of activation
- Sigmoid, ReLU, etc….
- Usually we use ReLU as first try on building neural network.

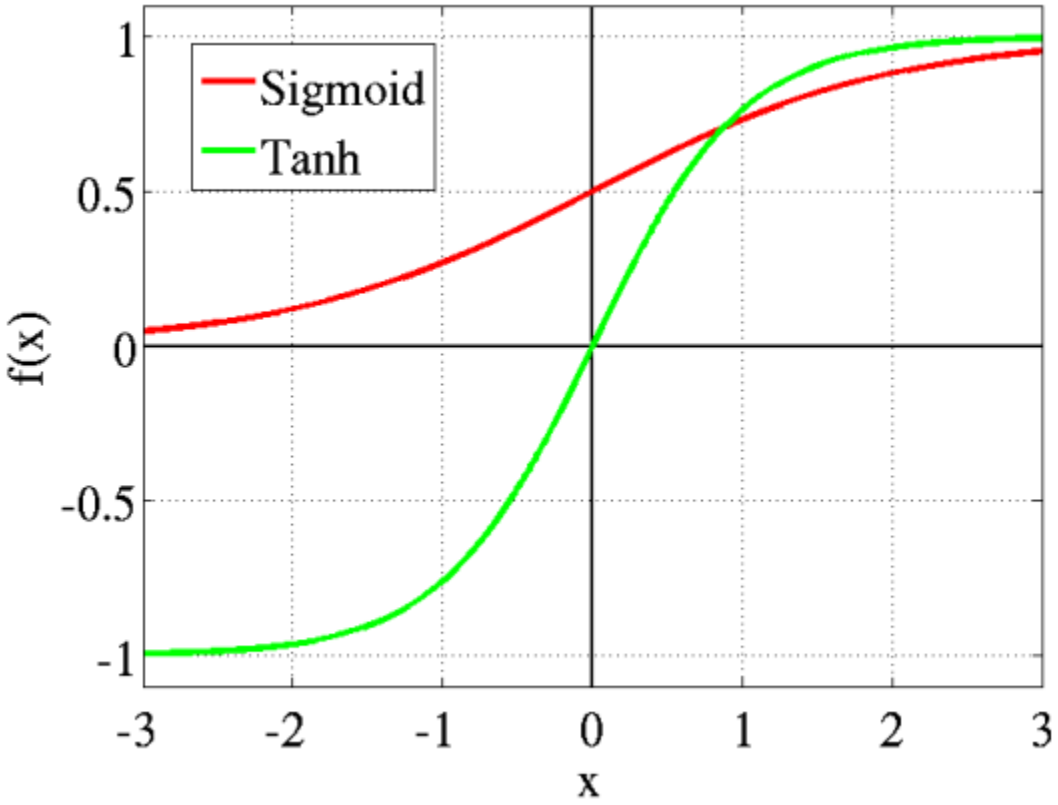
Sigmoid

- The sigmoid function exists between **(0 to 1).**
- Therefore, it is especially used for models where we have to **predict the probability** as an output.
- Since probability of anything exists only between the range of **0 and 1,** sigmoid is the right choice.



Tanh

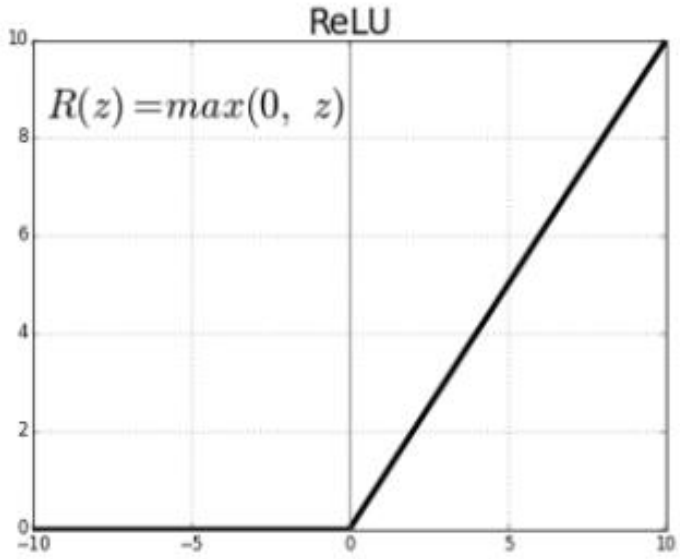
- Tanh is also like logistic sigmoid but better.
- The range of the tanh function is from (-1 to 1).



ReLU

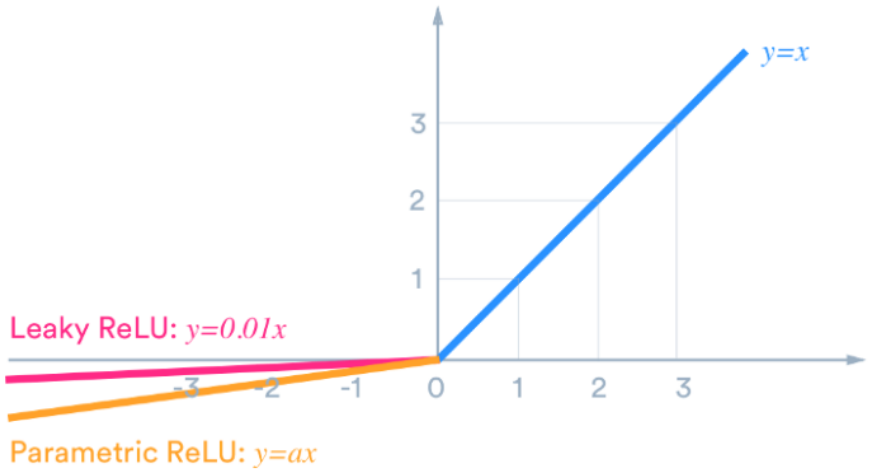
- ReLU is half rectified (from bottom).
- The ReLU is used in almost all the convolutional neural networks or deep learning.
  - Fast to compute.
  - Solve vanishing gradient problem.

$$y = \begin{cases} x, & if\ x \geq 0 \\ 0, & if\ x < 0 \end{cases}$$



Leaky ReLU

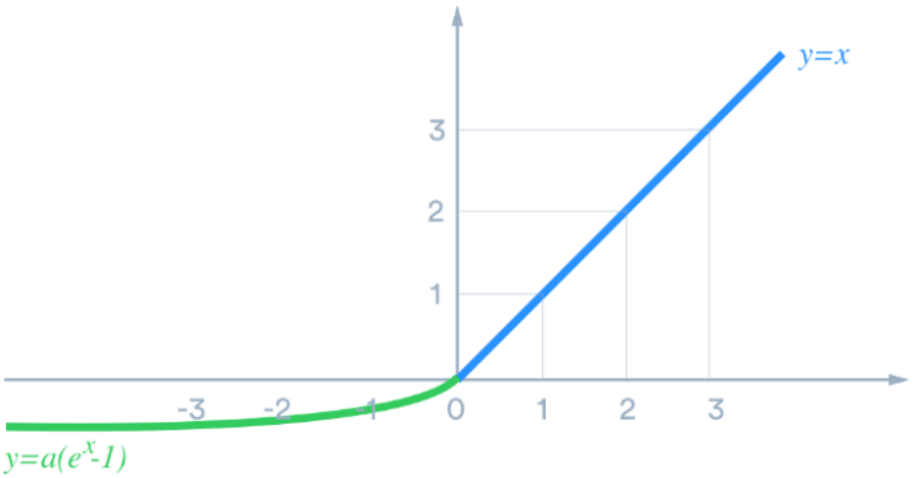
- Solve dying ReLU problem.
  - A ReLU neuron is “dead” if it’s stuck in the negative side and always outputs 0.
  - Once a neuron gets negative, it’s unlikely for it to recover.
- The leak helps to increase the range of the ReLU function. Usually, the value of **a** is 0.01 or so.
- When **a is not 0.01** then it is called **Parametric ReLU**.
- Therefore the **range** of the Leaky ReLU is (-infinity to infinity).



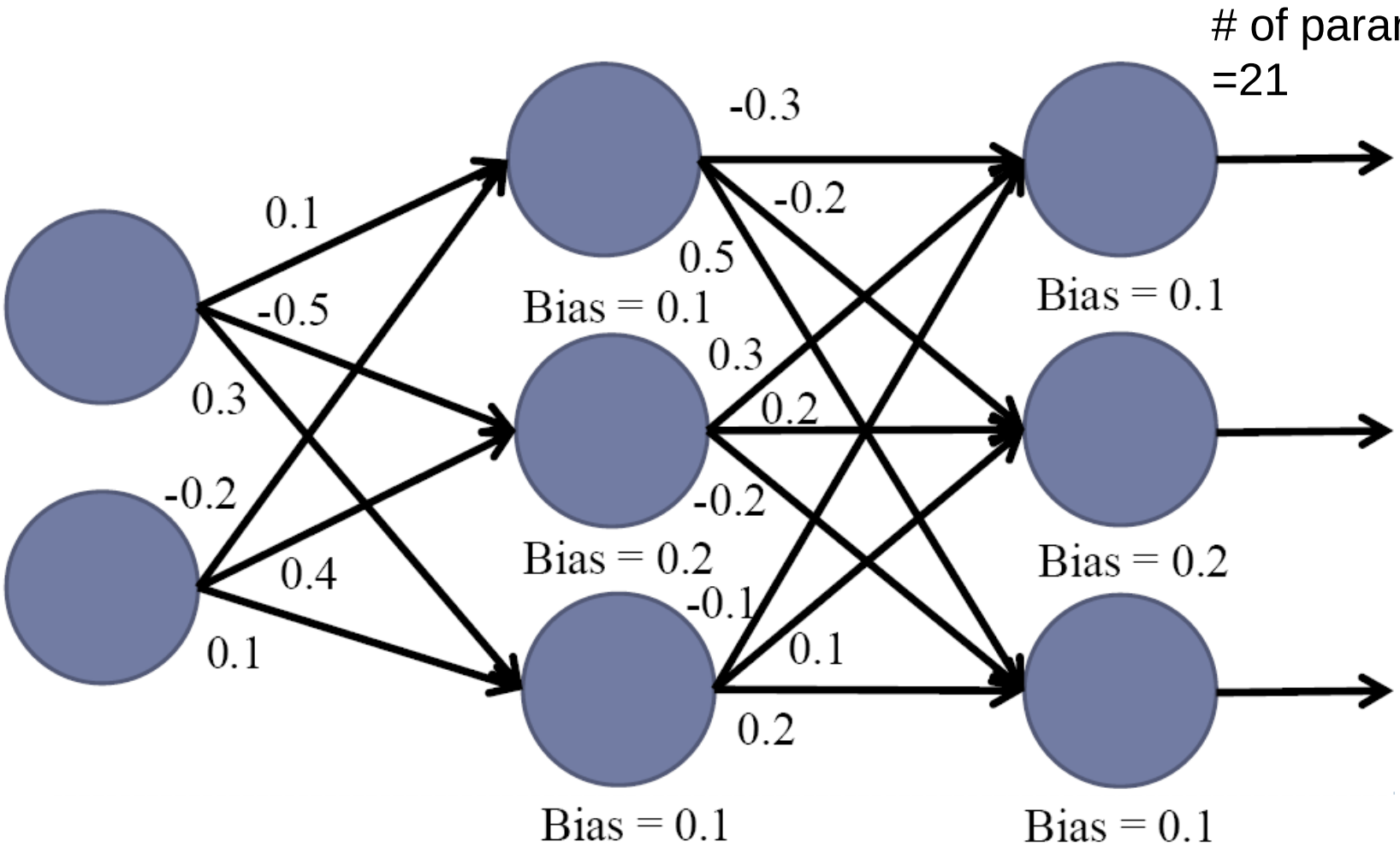
ELU

- Exponential Linear
- Similar to leaky ReLU, ELU has a small slope for negative values.
- Instead of a straight line, it uses a log curve.

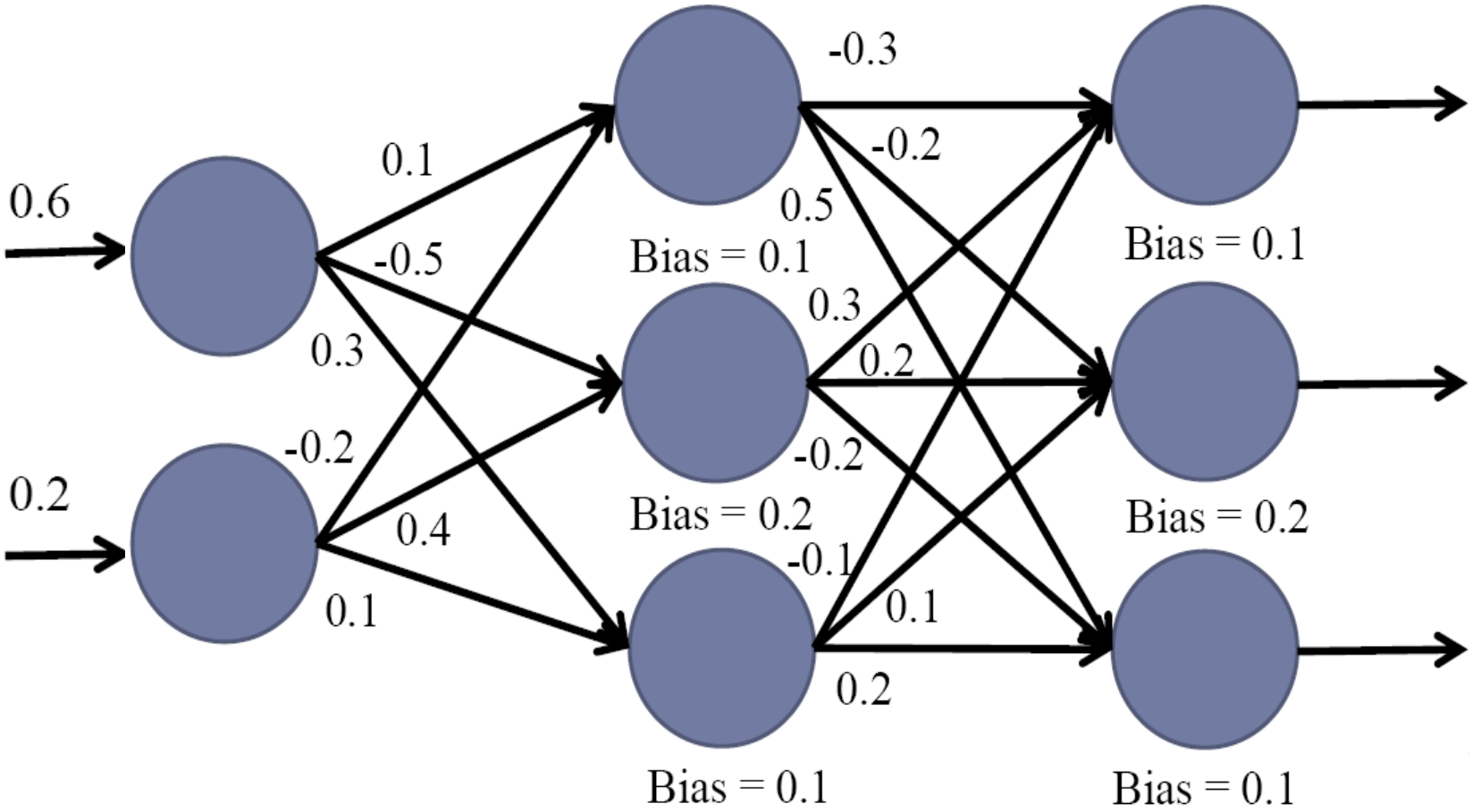
$$f(x) = \begin{cases} x, & if\ x > 0 \\ \alpha(\exp(x) - 1), & if\ x \leq 0 \end{cases}$$



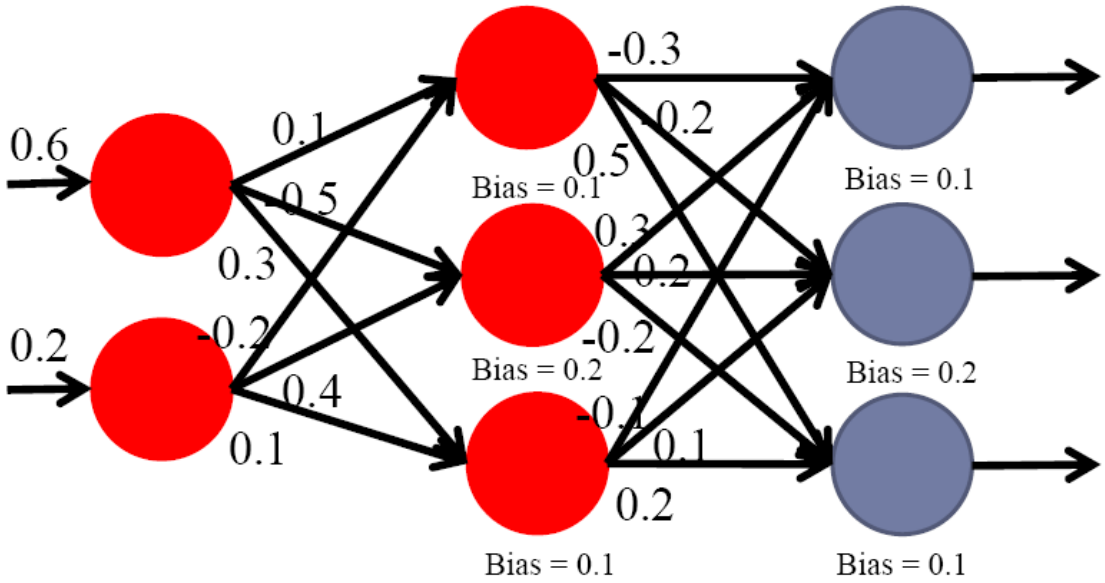
Example-Initialization



Example-Feed Data



Example-Forward Pass

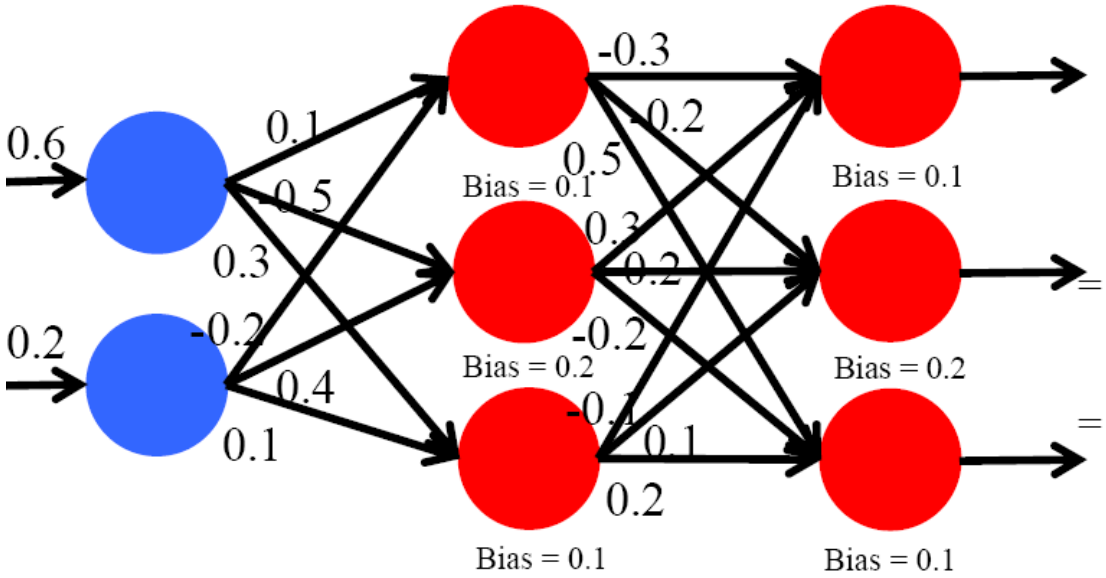


$$relu\left(\begin{bmatrix} 0.1 & -0.2 \\ -0.5 & 0.4 \\ 0.3 & 0.1 \end{bmatrix}\begin{bmatrix} 0.6 \\ 0.2 \end{bmatrix} + \begin{bmatrix} 0.1 \\ 0.2 \\ 0.1 \end{bmatrix}\right)$$

$$= relu\left(\begin{bmatrix} 0.12 \\ -0.02 \\ 0.3 \end{bmatrix}\right)$$

$$= \begin{bmatrix} 0.12 \\ 0 \\ 0.3 \end{bmatrix}$$

Example-Forward Pass



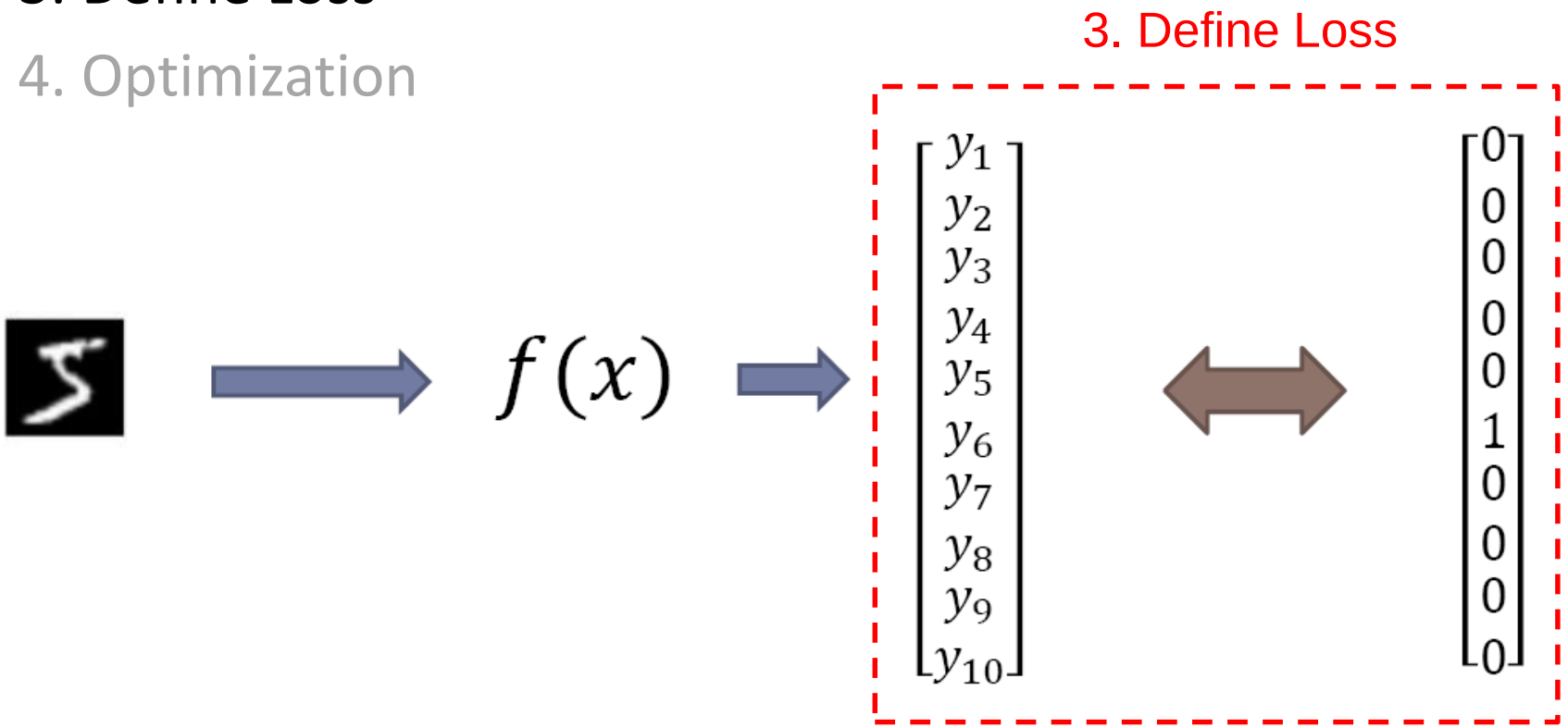
$$relu\left(\begin{bmatrix}-0.3 & 0.3 & -0.1\\ -0.2 & 0.2 & 0.1\\ 0.5 & -0.2 & 0.2\end{bmatrix}\begin{bmatrix}0.12\\ 0\\ 0.3\end{bmatrix}+\begin{bmatrix}0.1\\ 0.2\\ 0.1\end{bmatrix}\right)$$

$$= relu\left(\begin{bmatrix}-0.066\\ 0.006\\ 0.12\end{bmatrix}\right)$$

$$= \begin{bmatrix}0\\ 0.006\\ 0.12\end{bmatrix}$$

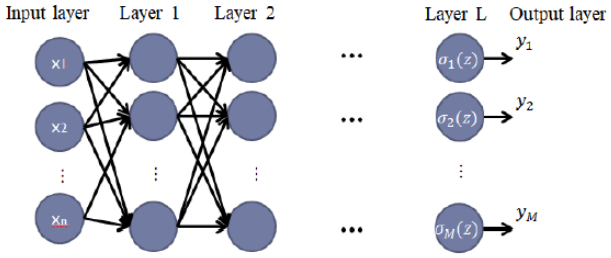
Steps

- 1. Prepare Dataset
- 2. Build DNN Model
- 3. Define Loss
- 4. Optimization



Loss Function

- There are many kinds of loss function
- Usually, it is a function that map multi-variables to a single value.
- We will introduce two loss function in DNN
  - Mean square
  - Cross-entropy



$$\begin{bmatrix} 0.15 \\ 0.2 \\ 0.3 \\ \vdots \end{bmatrix} \begin{matrix} \text{“0”} \\ \text{“1”} \\ \text{“2”} \end{matrix}$$

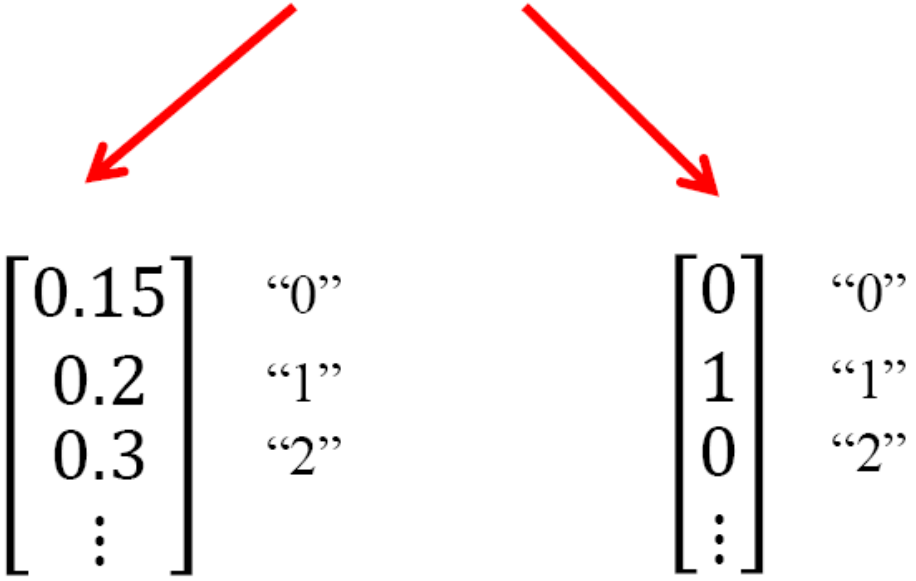
Output of NN look like this

$$\begin{bmatrix} 0 \\ 1 \\ 0 \\ \vdots \end{bmatrix} \begin{matrix} \text{“0”} \\ \text{“1”} \\ \text{“2”} \end{matrix}$$

What we want (one hot encode)

Mean Square

$$(y - \hat{y})^2$$



Recall: Information

- Information

$log\left(\frac{1}{p_i}\right)$, where pi is probability of an event.

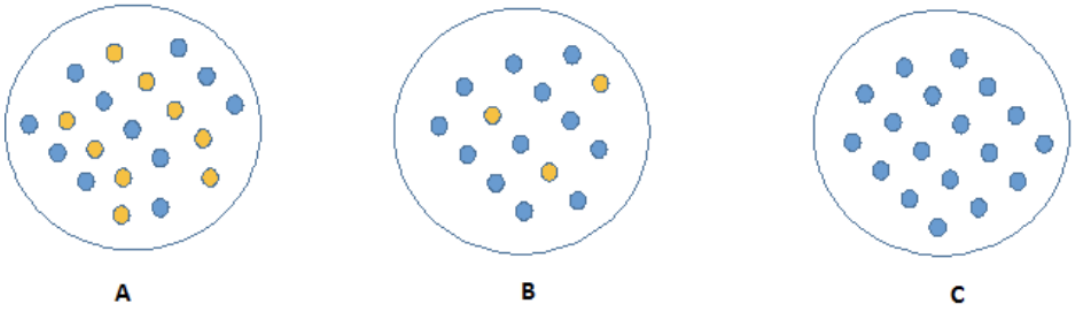
Sun rises in the east tomorrow	It will rain tomorrow in Taiwan

Which is more informative?

Recall: Entropy

- Entropy
  - Expected value(mean) of information contained in each message.
- Entropy can be seen as index of uncertainty.
  - Bigger mean more chaos.

$$H(y) = \sum_i y_i \log \frac{1}{y_i} = -\sum_i y_i \log y_i$$



Cross-Entropy

- Measurement on the difference between two probability distribution.
- Different distribution apply on entropy.
- Cross-entropy is grater than entropy.

$$H(y) = \sum_i y_i \log \frac{1}{y_i} = -\sum_i y_i \log y_i$$

Entropy

$$\boldsymbol{H(y) = -\sum_i y_i log \hat{y}_i}$$

Cross-entropy

Example (1/2)

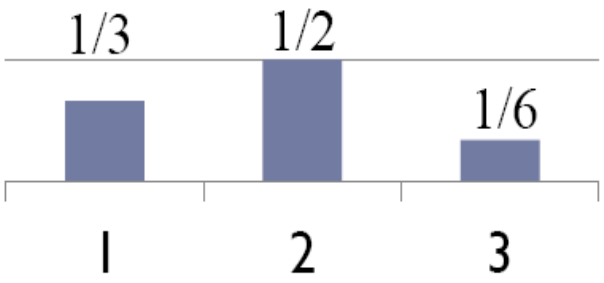
$$H(y) = \sum_i y_i \log \frac{1}{y_i} = -\sum_i y_i \log y_i$$

Entropy

$$H(y) = -\sum_i y_i log \hat{y}_i$$

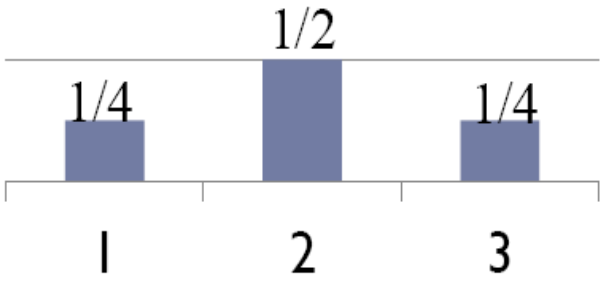
Cross-entropy

Probability distribution 1



Entropy on distribution 1
= 1/3 * log(3) + 1/2 * log(2) + 1/6 * log(6)

Probability distribution 2



Entropy on distribution 2
= 1/4 * log(4) + 1/2 * log(2) + 1/4 * log(4)

Cross-entropy on distribution 1 over distribution 2
= 1/3 * log(4) + 1/2 * log(2) + 1/6 * log(4)

Cross-entropy on distribution 2 over distribution 1
= 1/4 * log(3) + 1/2 * log(2) + 1/4 * log(6)

Example (2/2)

Entropy on distribution 1
$= 1/3 * \log(3) + 1/2 * \log(2) + 1/6 * \log(6)$
$= 0.439$

Entropy on distribution 2
$= 1/4 * \log(4) + 1/2 * \log(2) + 1/4 * \log(4)$
$=0.452$

Cross-entropy on distribution 1 over distribution 2
$= 1/3 * \log(4) + 1/2 * \log(2) + 1/6 * \log(4) = 0.456$

Cross-entropy on distribution 2 over distribution 1
$= 1/4 * \log(3) + 1/2 * \log(2) + 1/4 * \log(6) = 0.464$

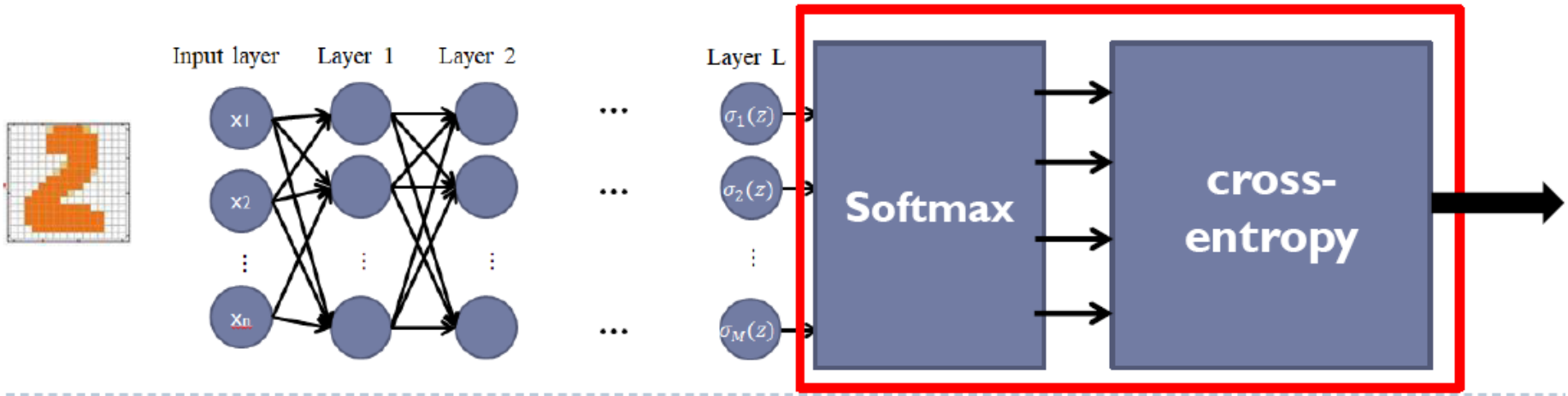
- Cross-entropy is greater than entropy
  Cross-entropy on distribution 1 over 2 > Entropy on distribution 1
  Cross-entropy on distribution 2 over 1 > Entropy on distribution 2

- If two distribution become closer
  - Value of cross-entropy is **closer** to entropy.

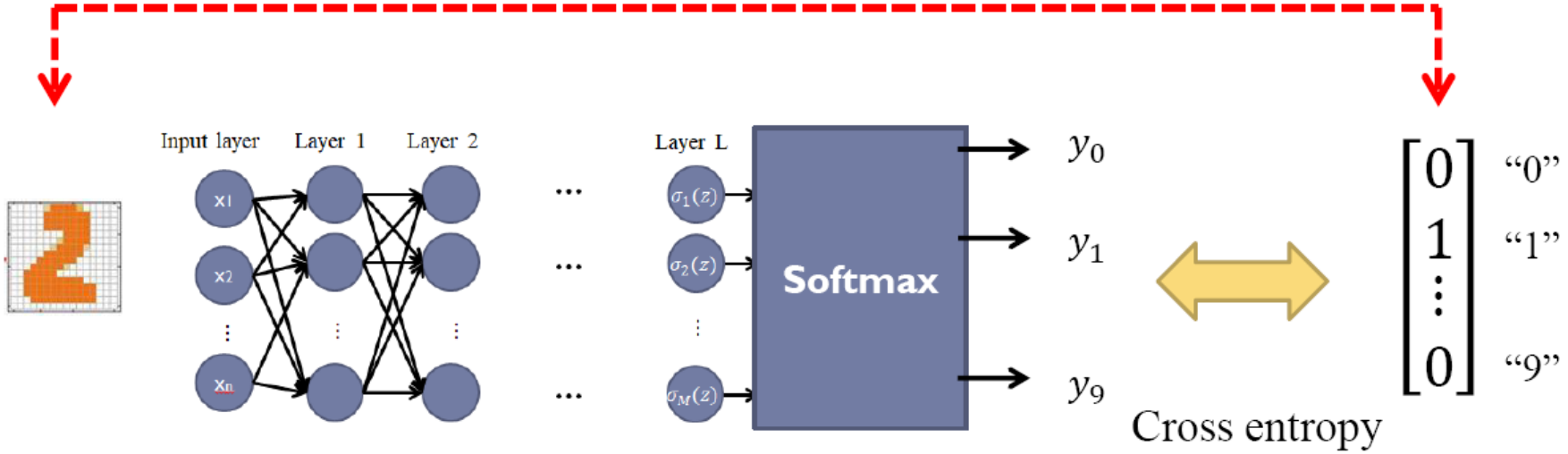
Cross-entropy with Softmax

- Cross-entropy usually come with softmax layer in NN.
- Softmax function squash all of elements in vector to [0,1].

$$\sigma(\mathbf{z})_j = \frac{e^{z_j}}{\sum_{k=1}^{K} e^{z_k}} \quad \text{for } j = 1, \ldots, K.$$



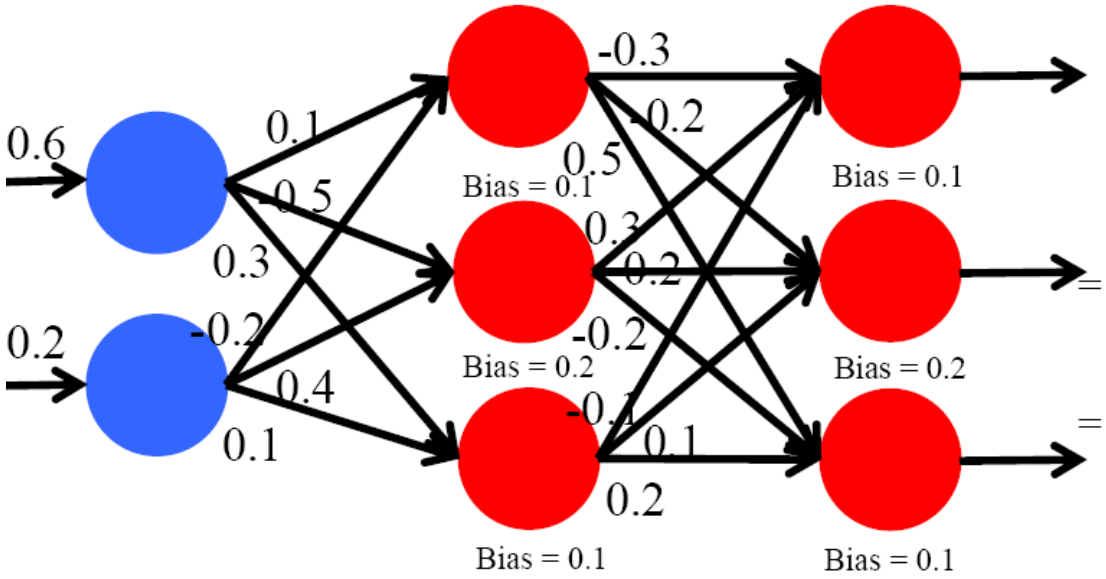
Cross-entropy with Softmax



Loss/Cost $= 0 * y_0 + 1 * y_1 + 0 * y_2 + \cdots + 0 * y_9$

$$H(y) = -\sum_i y_i log\hat{y}_i$$

Example-Forward Pass

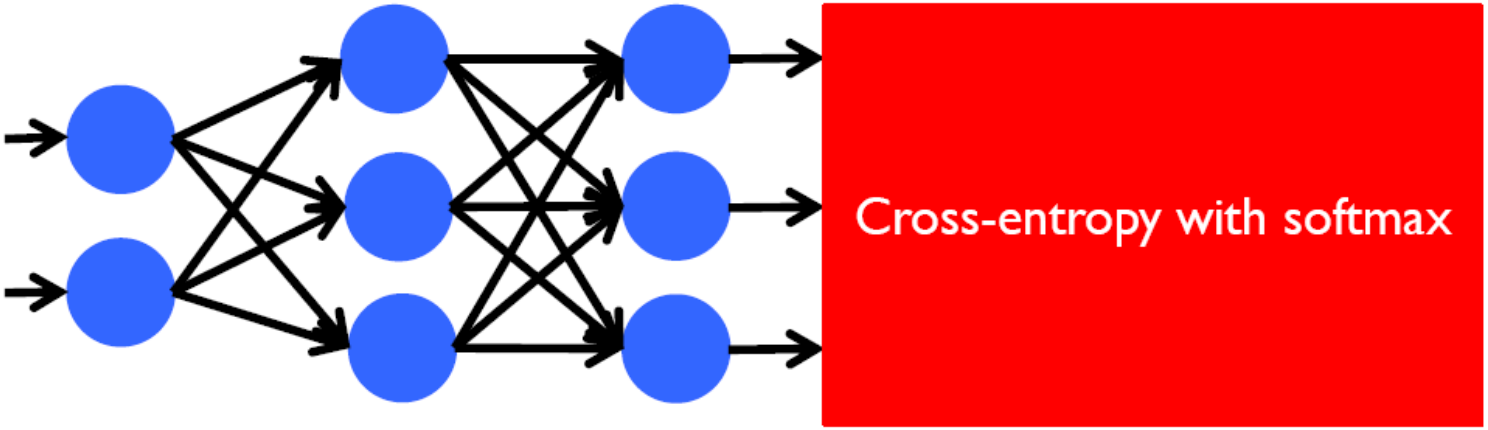


$$relu\left(\begin{bmatrix}-0.3 & 0.3 & -0.1\\ -0.2 & 0.2 & 0.1\\ 0.5 & -0.2 & 0.2\end{bmatrix}\begin{bmatrix}0.12\\ 0\\ 0.3\end{bmatrix}+\begin{bmatrix}0.1\\ 0.2\\ 0.1\end{bmatrix}\right)$$

$$= relu\left(\begin{bmatrix}-0.066\\ 0.006\\ 0.12\end{bmatrix}\right)$$

$$= \begin{bmatrix}0\\ 0.006\\ 0.12\end{bmatrix}$$

Example-Forward Pass

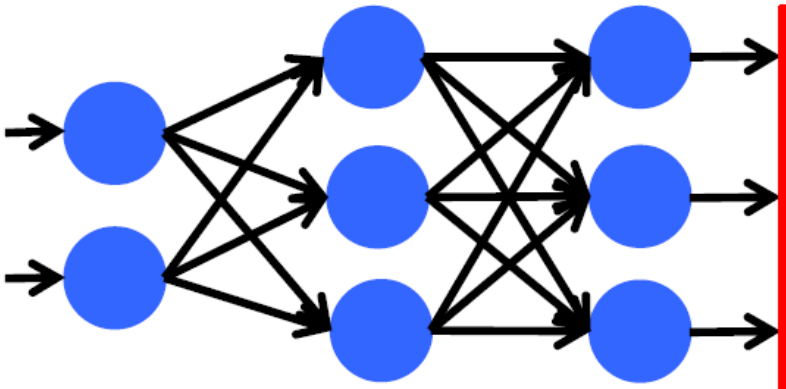


$$\text{Softmax}\left(\begin{bmatrix}0\\0.006\\0.12\end{bmatrix}\right) = \begin{bmatrix}0.319\\0.321\\0.36\end{bmatrix}$$

$$\sigma(\mathbf{z})_j = \frac{e^{z_j}}{\sum_{k=1}^{K} e^{z_k}} \quad \text{for } j = 1, \ldots, K.$$

$$\frac{e^{0}}{e^{0} + e^{0.006} + e^{0.12}} = 0.319$$

Example-Forward Pass



Cross-entropy with softmax

What we expect (label)

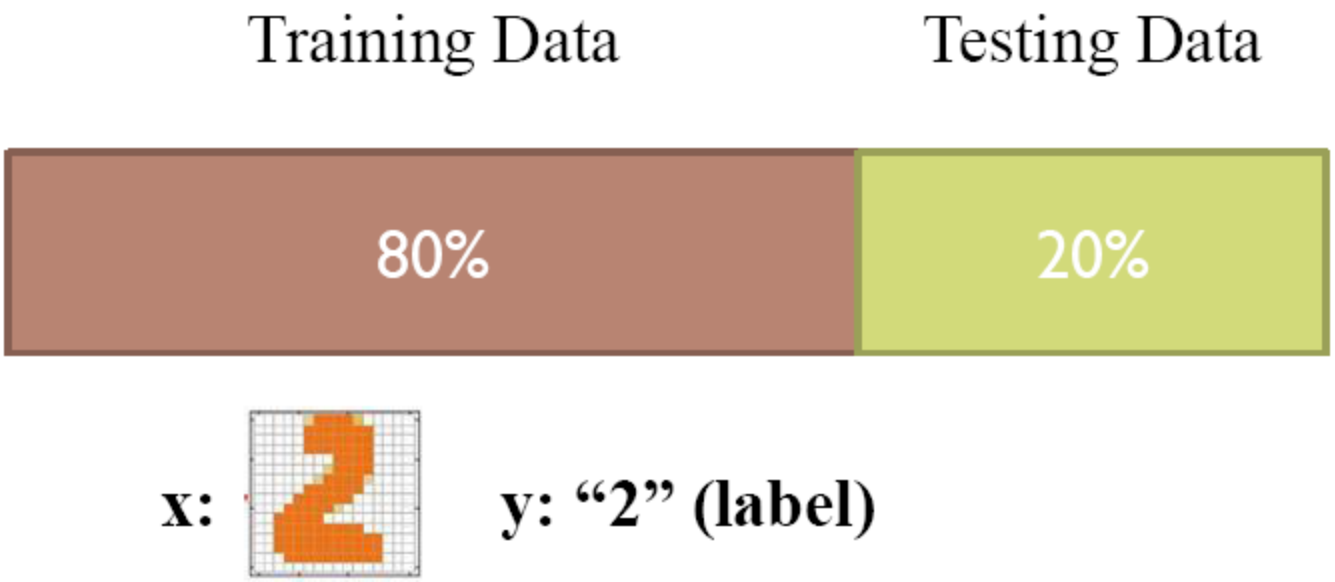
$$\begin{bmatrix} 0 \\ 1 \\ 0 \end{bmatrix}$$

- 0 * ln(0.319) – 1 * ln(0.321) – 0 * ln(0.36)
= 1.1363

$$-\sum_{i=0}^{class\ \#} \hat{y}_i \ln(y_i)$$

- This is large at the beginning. During training, this should decrease.

How to Feed Data (1/5)

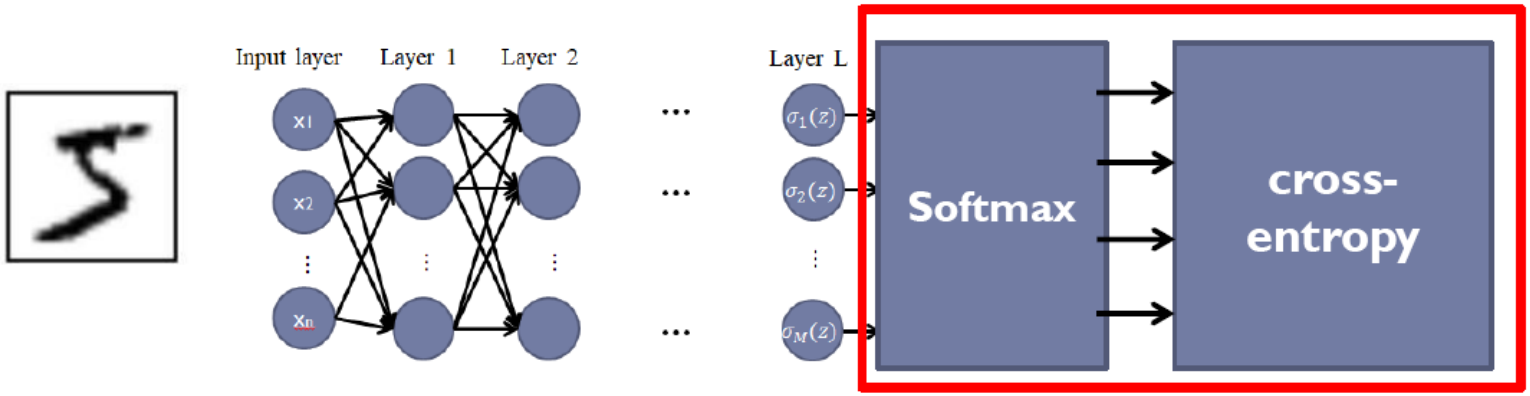


How to Feed Data (2/5)

How to feed data	Mean Square	Cross-entropy
All data at a time	$\frac{1}{N}\sum_{i=1}^{N}(y_i-\hat{y}_i)^2$	$\frac{1}{N}\sum_{j=1}^{N}(-\sum_{i=0}^{class\ \#}\hat{y}_i\ln(y_i))$
One data at a time	$(y-\hat{y})$	$-\sum_{i=0}^{class\ \#}\hat{y}_i\ln(y_i)$
Batch of data a time (B < N)	$\frac{1}{B}\sum_{i=1}^{B}(y_i-\hat{y}_i)^2$	$\frac{1}{B}\sum_{j=1}^{B}(-\sum_{i=0}^{class\ \#}\hat{y}_i\ln(y_i))$

How to Feed Data (3/5)

Pick one training data at a time

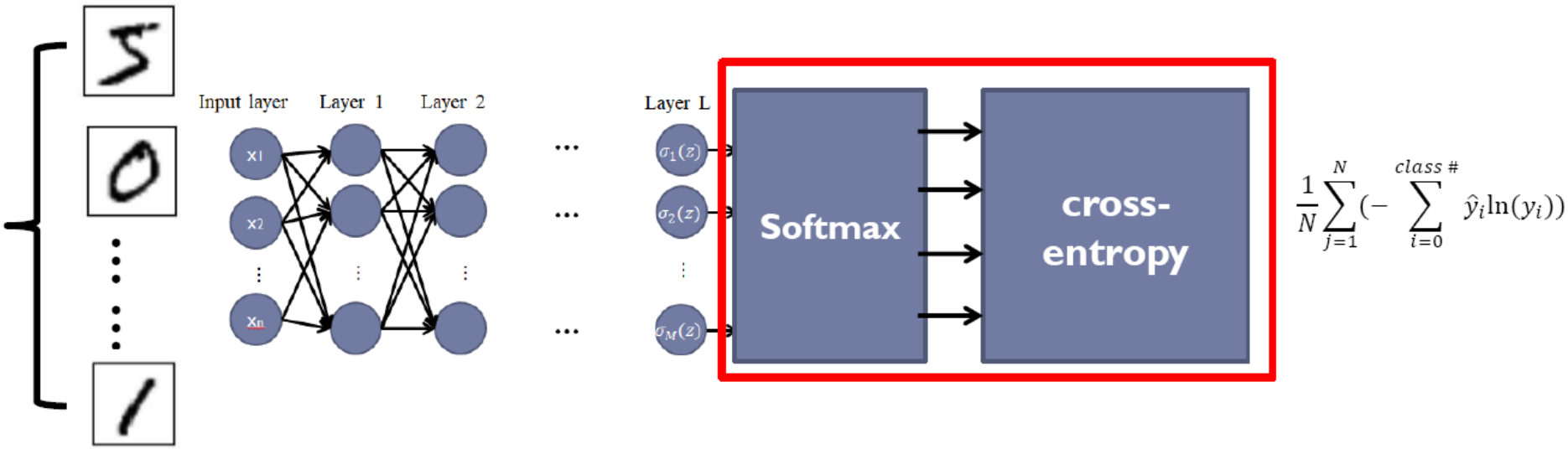


$$-\sum_{i=0}^{class\ \#} \hat{y}_i \ln(y_i)$$

How to Feed Data (4/5)

Pick all training data at a time

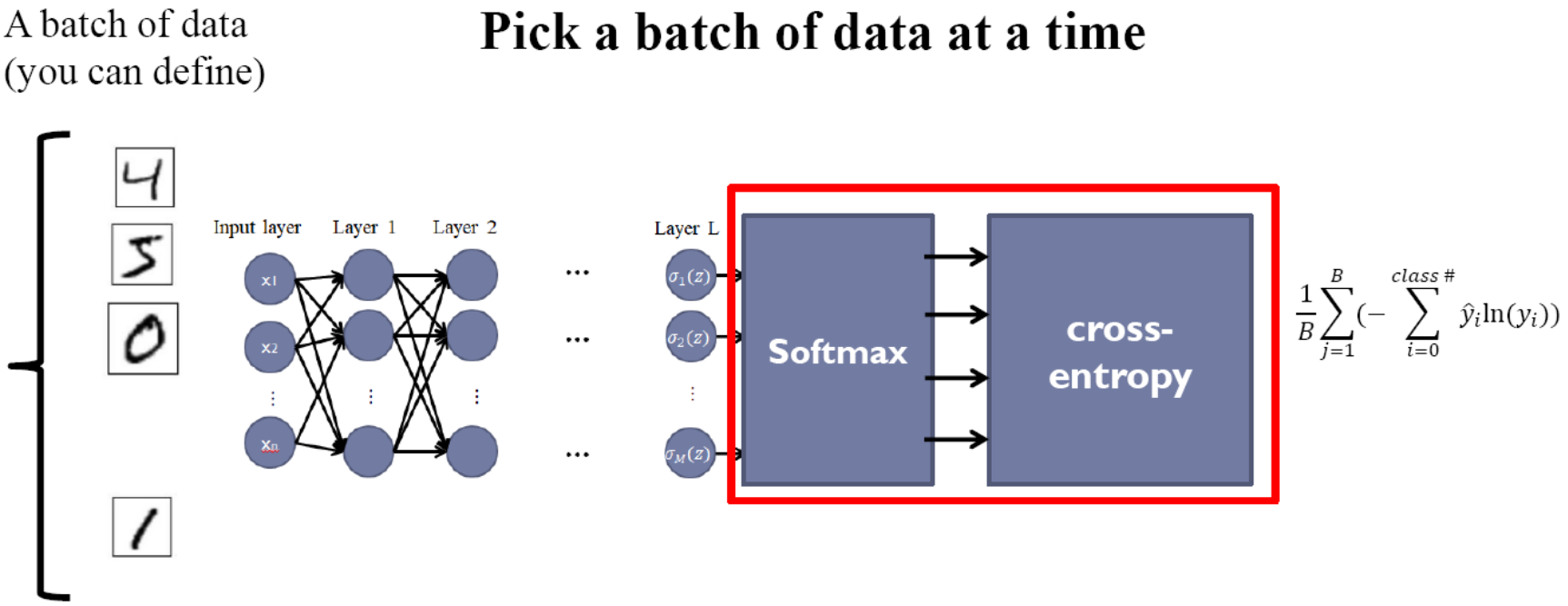
All training data



$$\frac{1}{N}\sum_{j=1}^{N}(-\sum_{i=0}^{class\ \#}\hat{y}_i \ln(y_i))$$

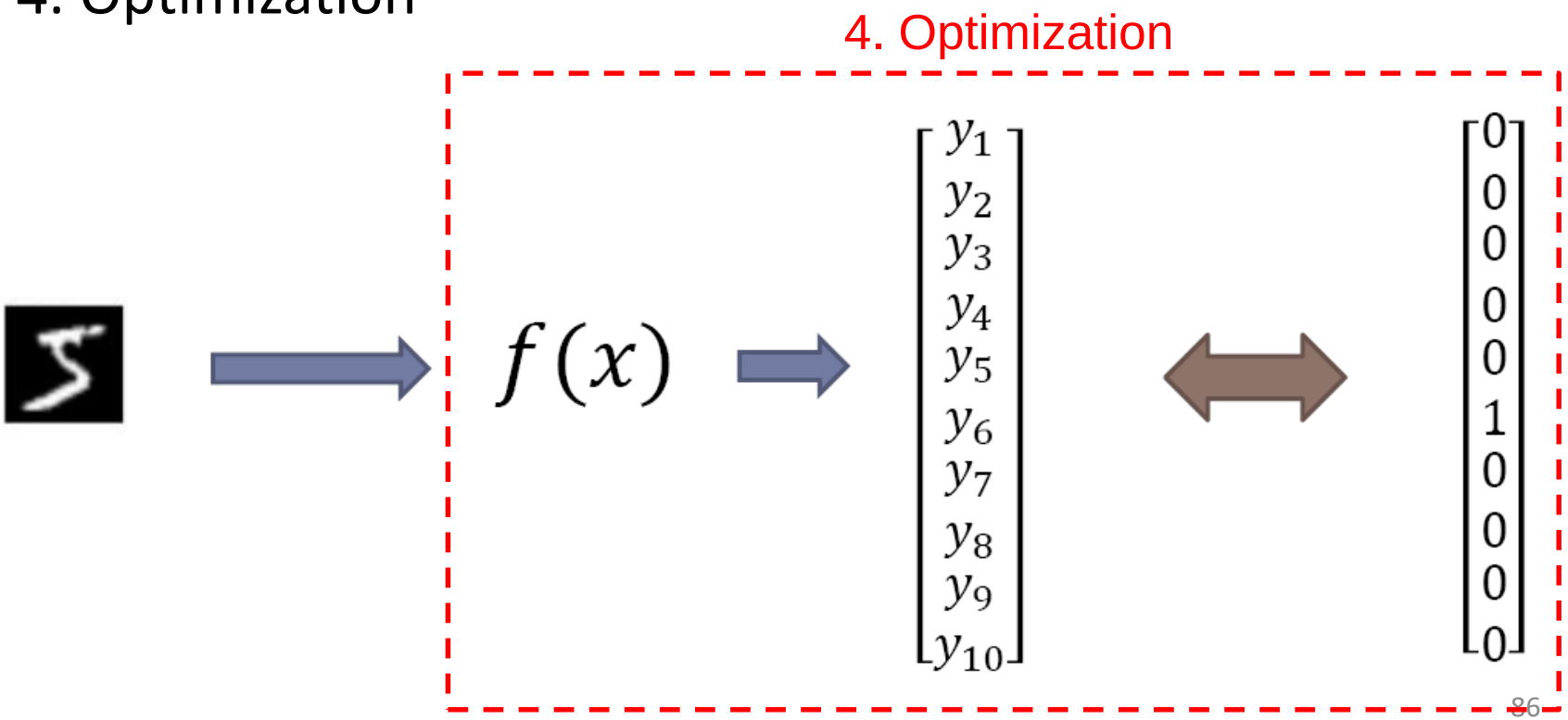
How to Feed Data (5/5)

- Batch size is usually set multiples of 2.
- There is not best answer. It is a process of try and error.



Steps

- 1. Prepare Dataset
- 2. Build DNN Model
- 3. Define Loss
- 4. Optimization



Optimization

- Gradient decent
- Increase the performance of optimization
  - Tips 1:Tune Learning Rate
  - Tips2: Stochastic Gradient Descent
  - Tips3: Feature Scaling
- How to update many parameters

Best Function = Best Parameters

$$y = f(x) = \sigma\left(W^L \ldots \sigma\left(W^2 \sigma\left(W^1 x + b^1\right) + b^1\right) \ldots + b^L\right)$$

function set

because different parameters W and b lead to different function

Formal way to define a function set:

$f(x;\theta)$ → parameter set

$$\theta = \left\{W^1, b^1, W^2, b^2 \cdots W^L, b^L\right\}$$

Pick the "best" function f*

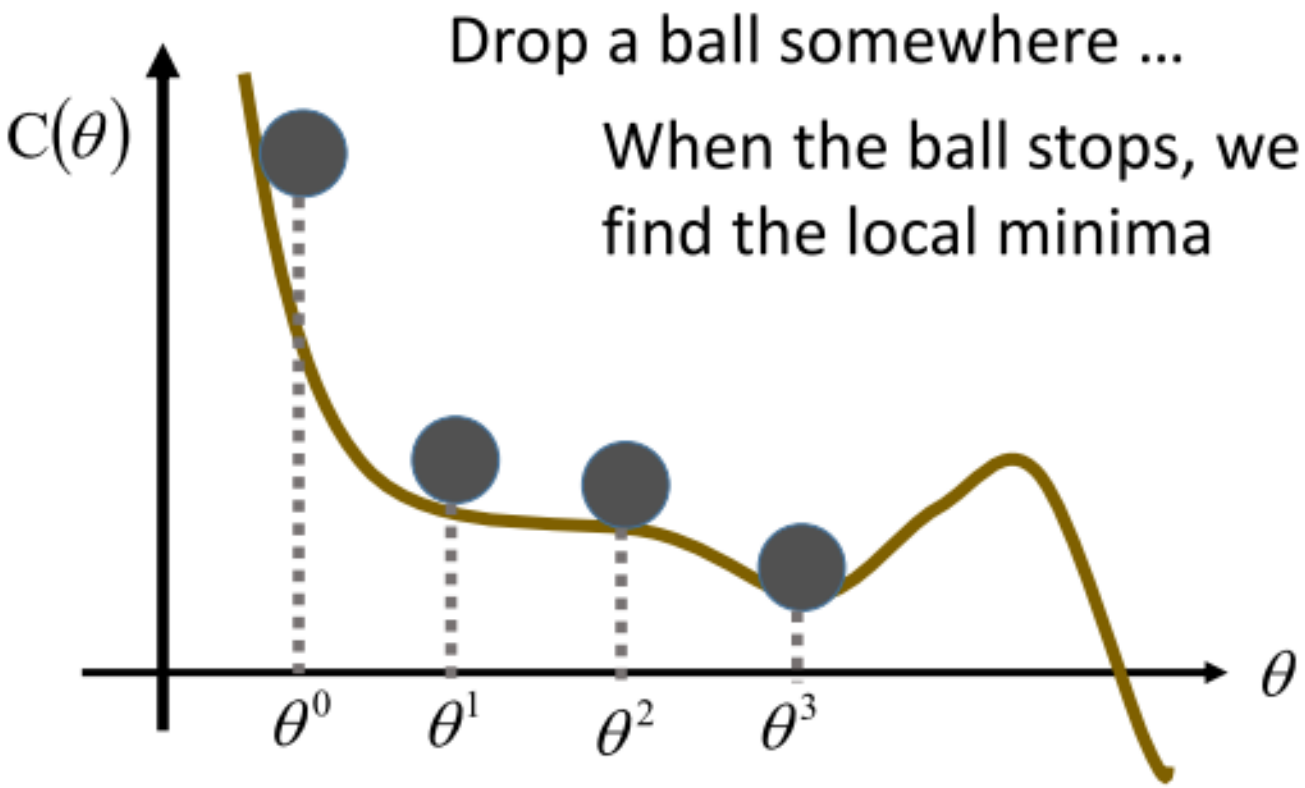
Pick the "best" parameter set θ*

Statement of Problems

- Statement of problems:
  - There is a function C(θ)
    - θ represents parameter set
    - $\theta = \{\theta_1, \theta_2, \theta_3, \ldots\ldots\}$
  - Find θ^* that minimizes C(θ)
- Brute force?
  - Enumerate all possible θ
- Calculus?
  - Find θ^* such that $\left.\frac{\partial C(\theta)}{\partial \theta_1}\right|_{\theta=\theta^*} = 0, \left.\frac{\partial C(\theta)}{\partial \theta_2}\right|_{\theta=\theta^*} = 0, \ldots$

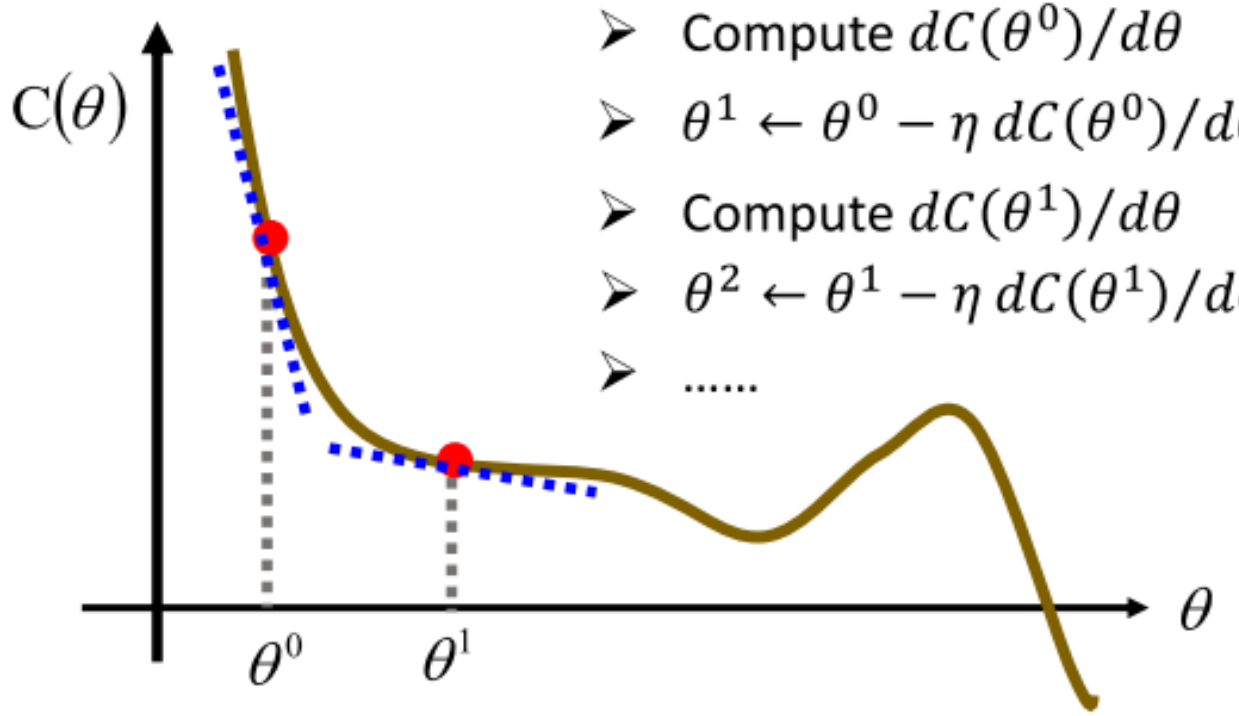
Gradient Descent

- For simplification, first consider that θ has only one variable



Gradient Descent

- For simplification, first consider that θ has only one variable



- Randomly start at θ^0
- Compute $dC(\theta^0)/d\theta$
- $\theta^1 \leftarrow \theta^0 - \eta\, dC(\theta^0)/d\theta$
- Compute $dC(\theta^1)/d\theta$
- $\theta^2 \leftarrow \theta^1 - \eta\, dC(\theta^1)/d\theta$
- ……

η is called *"learning rate"*

Formal Derivation of Gradient Descent (1/2)

Suppose that θ has two variables $\{\theta_1, \theta_2\}$

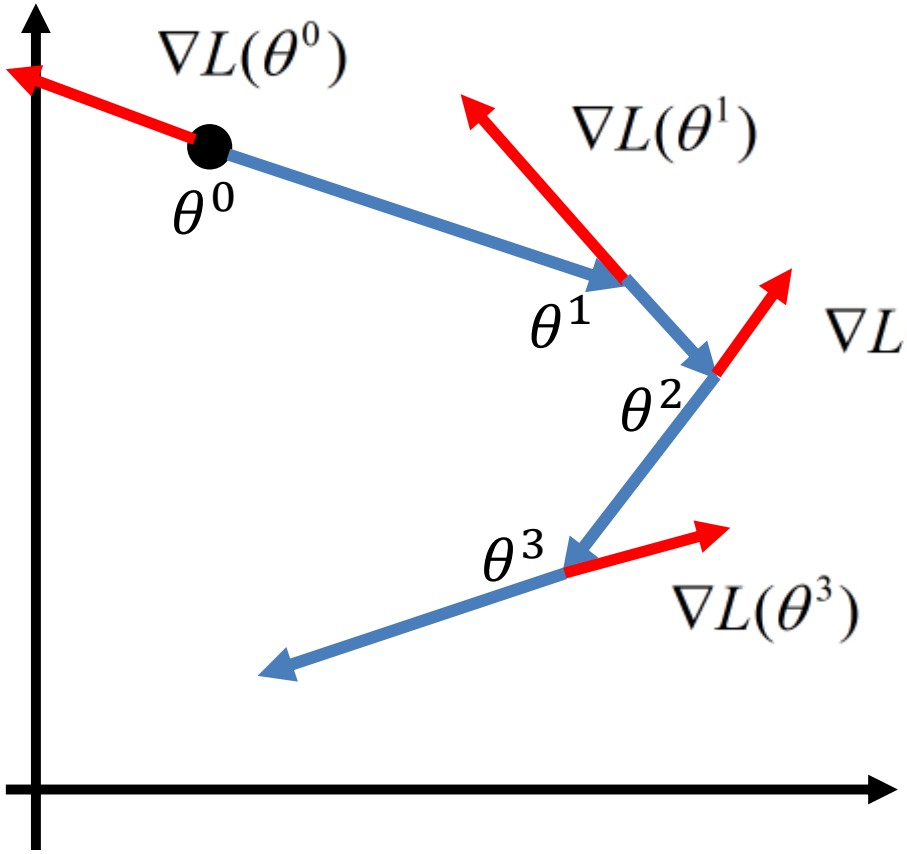
Randomly start at $\theta^0 = \begin{bmatrix} \theta_1^0 \\ \theta_2^0 \end{bmatrix}$

$$\nabla L(\theta) = \begin{bmatrix} \partial L(\theta_1)/\partial\theta_1 \\ \partial L(\theta_2)/\partial\theta_2 \end{bmatrix}$$

$$\begin{bmatrix} \theta_1^1 \\ \theta_2^1 \end{bmatrix} = \begin{bmatrix} \theta_1^0 \\ \theta_2^0 \end{bmatrix} - \eta \begin{bmatrix} \partial L(\theta_1^0)/\partial\theta_1 \\ \partial L(\theta_2^0)/\partial\theta_2 \end{bmatrix} \Rightarrow \theta^1 = \theta^0 - \eta\nabla L(\theta^0)$$

$$\begin{bmatrix} \theta_1^2 \\ \theta_2^2 \end{bmatrix} = \begin{bmatrix} \theta_1^1 \\ \theta_2^1 \end{bmatrix} - \eta \begin{bmatrix} \partial L(\theta_1^1)/\partial\theta_1 \\ \partial L(\theta_2^1)/\partial\theta_2 \end{bmatrix} \Rightarrow \theta^2 = \theta^1 - \eta\nabla L(\theta^1)$$

Formal Derivation of Gradient Descent (2/2)



Start at position θ^0

Compute gradient at θ^0

Move to $\theta^1 = \theta^0 - \eta\nabla L(\theta^0)$

Compute gradient at θ^1

Move to $\theta^2 = \theta^1 - \eta\nabla L(\theta^1)$

⋮

Gradient

Movement

Like Playing a Game

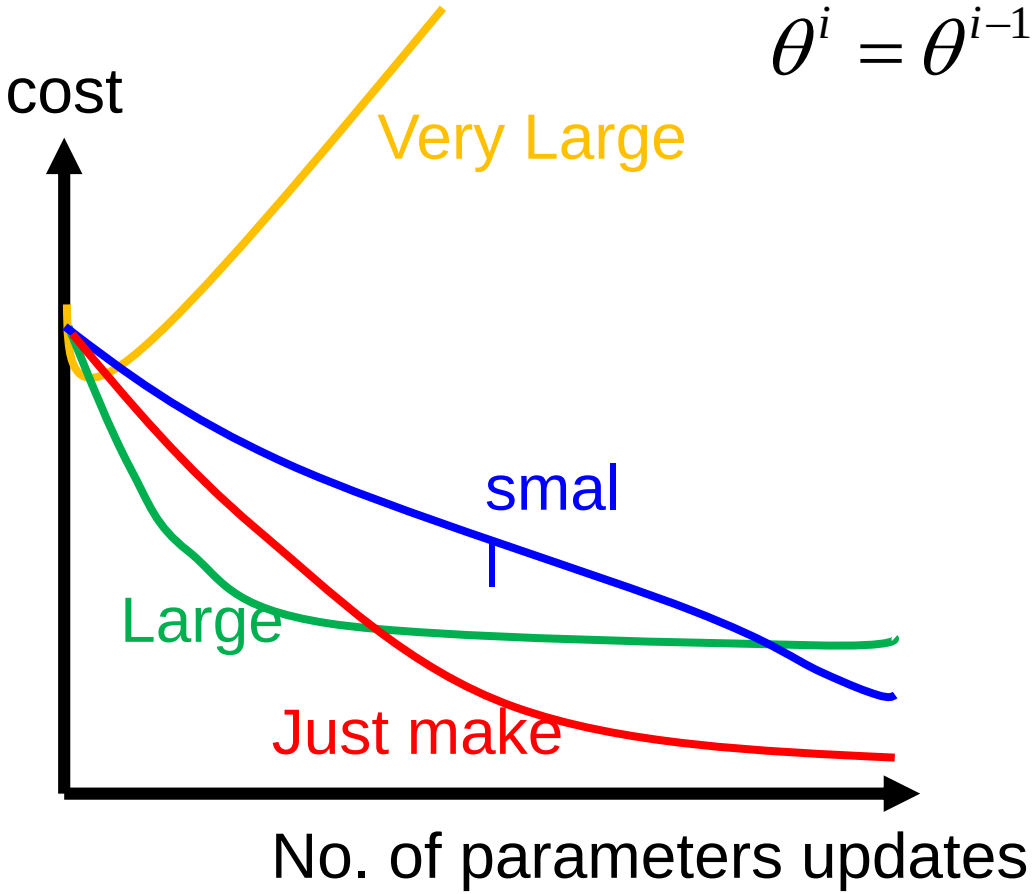
Optimization

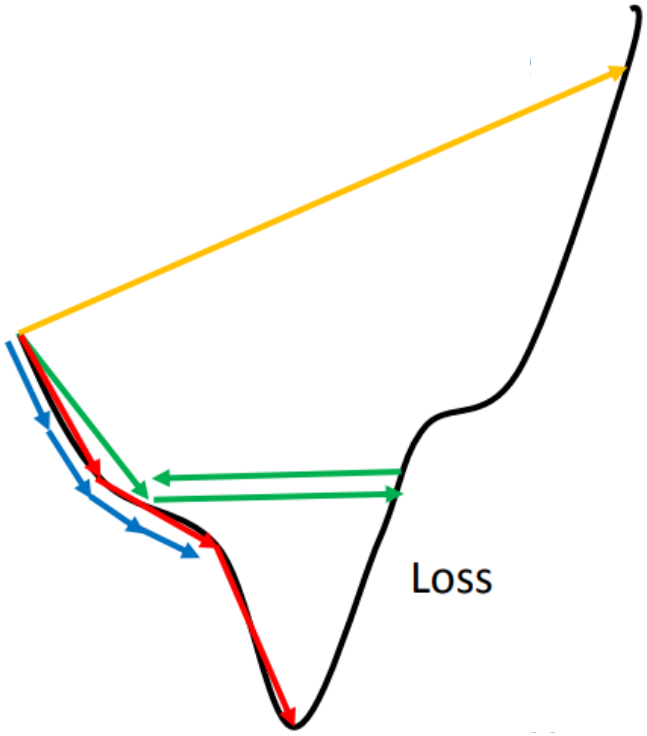
- Gradient decent
- Increase the performance of optimization
  - Tips 1:Tune Learning Rate
  - Tips2: Stochastic Gradient Descent
  - Tips3: Feature Scaling
- How to update many parameters

The Drawback of Gradient Decent

- Learning rate would not decay over time.
- Direction of learning rate is fixed.

$$\theta^i = \theta^{i-1} - \eta \nabla C\left(\theta^{i-1}\right)$$





Adaptive Learning Rate

- Popular and simple idea: Reduce the learning rate by some factor every few epochs.
  - At the beginning, we are far from the destination, so we use larger learning rate.
  - After several epochs, we are close to the destination, so we reduce the learning rate.
  - EX: 1/t decay: $\eta^t = \eta / \sqrt{t+1}$
- Learning rate cannot be one-size-fits-all
  - Give different parameters different learning rates.

Adagrad (1/3)

- Divide the learning rate of each parameter by the root mean square of its previous derivatives.

$$\eta^t = \frac{\eta}{\sqrt{t+1}} \qquad g^t = \frac{\partial L(\theta^t)}{\partial w}$$

Vanilla Gradient descent

$$w^{t+1} \leftarrow w^t - \eta^t g^t$$

w is one parameters

Adagrad

$$w^{t+1} \leftarrow w^t - \frac{\eta^t}{\sigma^t} g^t$$

σ^t: ***root mean square*** of the previous derivatives of parameter w

Parameter dependent

Adagrad (2/3)

$$\eta^t = \frac{\eta}{\sqrt{t+1}} \qquad g^t = \frac{\partial L(\theta^t)}{\partial w}$$

$$w^1 \leftarrow w^0 - \frac{\eta^0}{\sigma^0} g^0 \qquad \sigma^0 = \sqrt{(g^0)^2}$$

$$w^2 \leftarrow w^1 - \frac{\eta^1}{\sigma^1} g^1 \qquad \sigma^1 = \sqrt{\frac{1}{2}[(g^0)^2 + (g^1)^2]}$$

$$w^3 \leftarrow w^2 - \frac{\eta^2}{\sigma^2} g^2 \qquad \sigma^2 = \sqrt{\frac{1}{3}[(g^0)^2 + (g^1)^2 + (g^2)^2]}$$

$$\vdots$$

$$w^{t+1} \leftarrow w^t - \frac{\eta^t}{\sigma^t} g^t \qquad \sigma^t = \sqrt{\frac{1}{t+1}\sum_{i=0}^{t} (g^i)^2}$$

Adagrad (3/3)

- Divide the learning rate of each parameter by the root mean square of its previous derivatives.

$$w^{t+1} \leftarrow w^t - \frac{\eta^t}{\sigma^t} g^t$$

η^t: $\eta^t = \frac{\eta}{\sqrt{t+1}}$ 1/t decay

σ^t: $\sigma^t = \sqrt{\frac{1}{t+1}\sum_{i=0}^{t}(g^i)^2}$

$$w^{t+1} \leftarrow w^t - \frac{\eta}{\sqrt{\sum_{i=0}^{t}(g^i)^2}} g^t$$

Anything Weird?

$$\eta^t = \frac{\eta}{\sqrt{t+1}} \qquad g^t = \frac{\partial L(\theta^t)}{\partial w}$$

Vanilla Gradient descent

$$w^{t+1} \leftarrow w^t - \eta^t g^t$$

Larger gradient, larger step

Adagrad

$$w^{t+1} \leftarrow w^t - \frac{\eta}{\sqrt{\sum_{i=0}^{t}(g^i)^2}} g^t$$

Intuitive Reason

- How surprise it is 反差

特别大

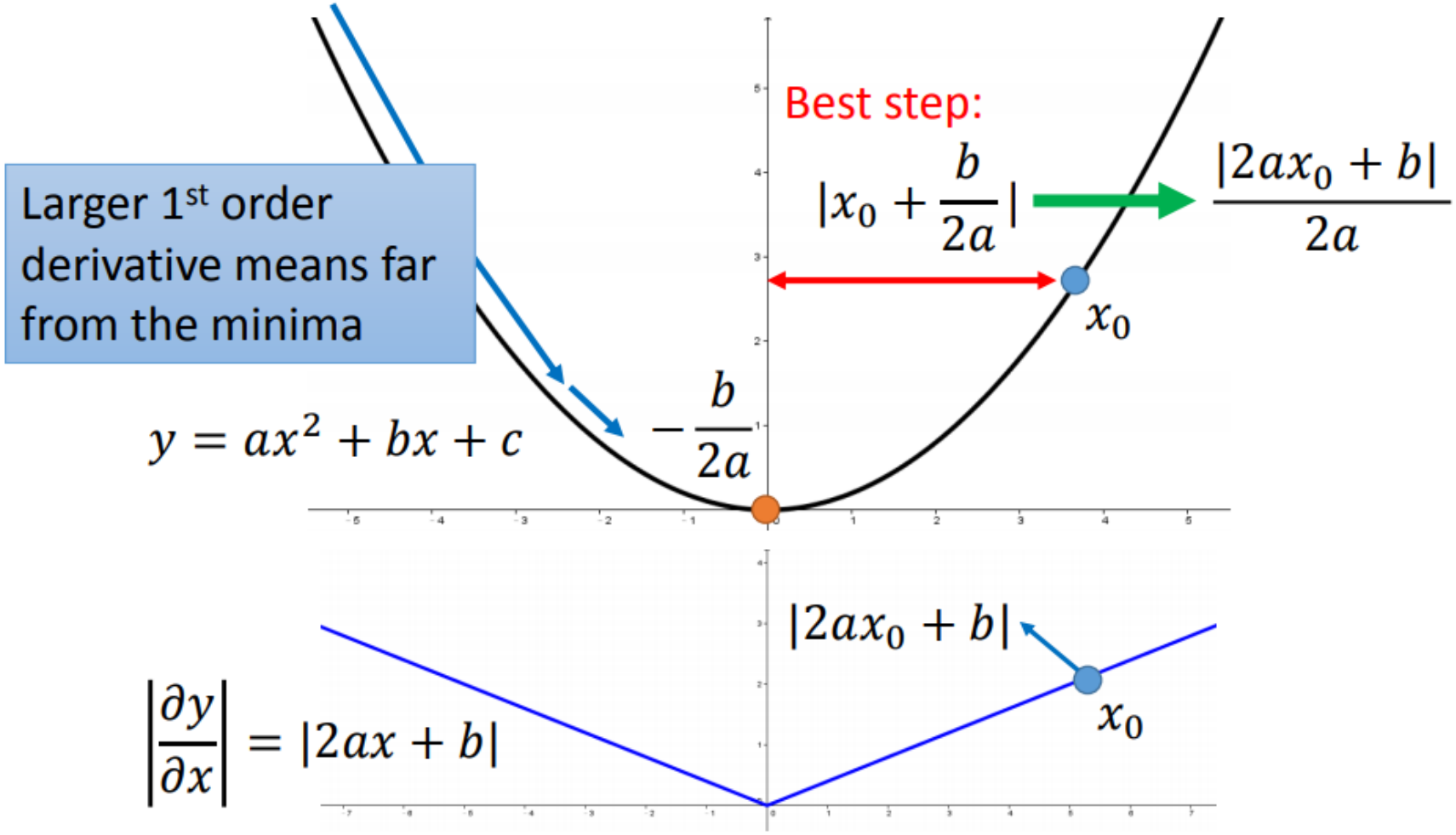
g^0	g^1	g^2	g^3	g^4	……
0.001	0.001	0.003	0.002	0.1	……
g^0	g^1	g^2	g^3	g^4	……
10.8	20.9	31.7	12.1	0.1	……

特别小

$$w^{t+1} \leftarrow w^t - \frac{\eta}{\sqrt{\sum_{i=0}^{t}(g^i)^2}} g^t$$

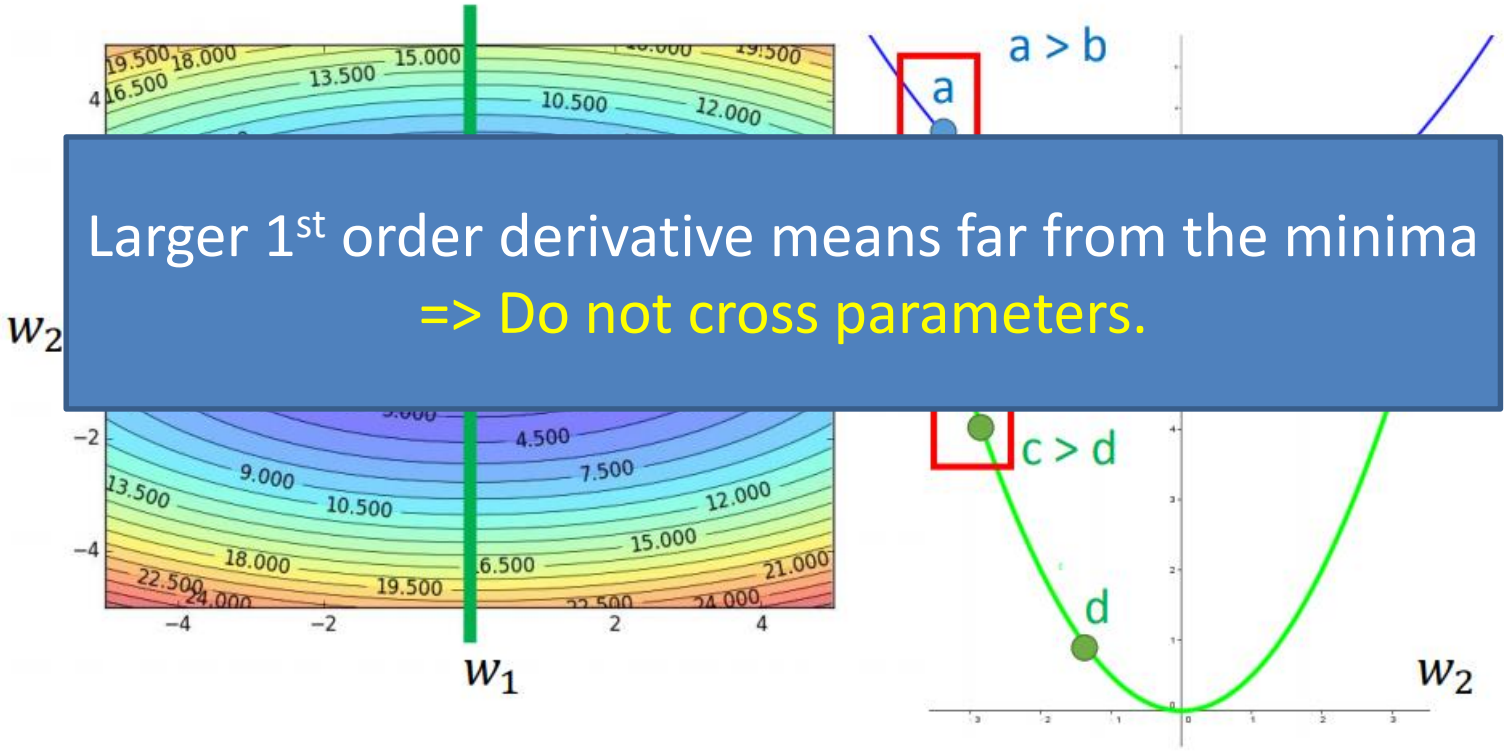
→ 造成反差的效果

Larger gradient, Larger steps?

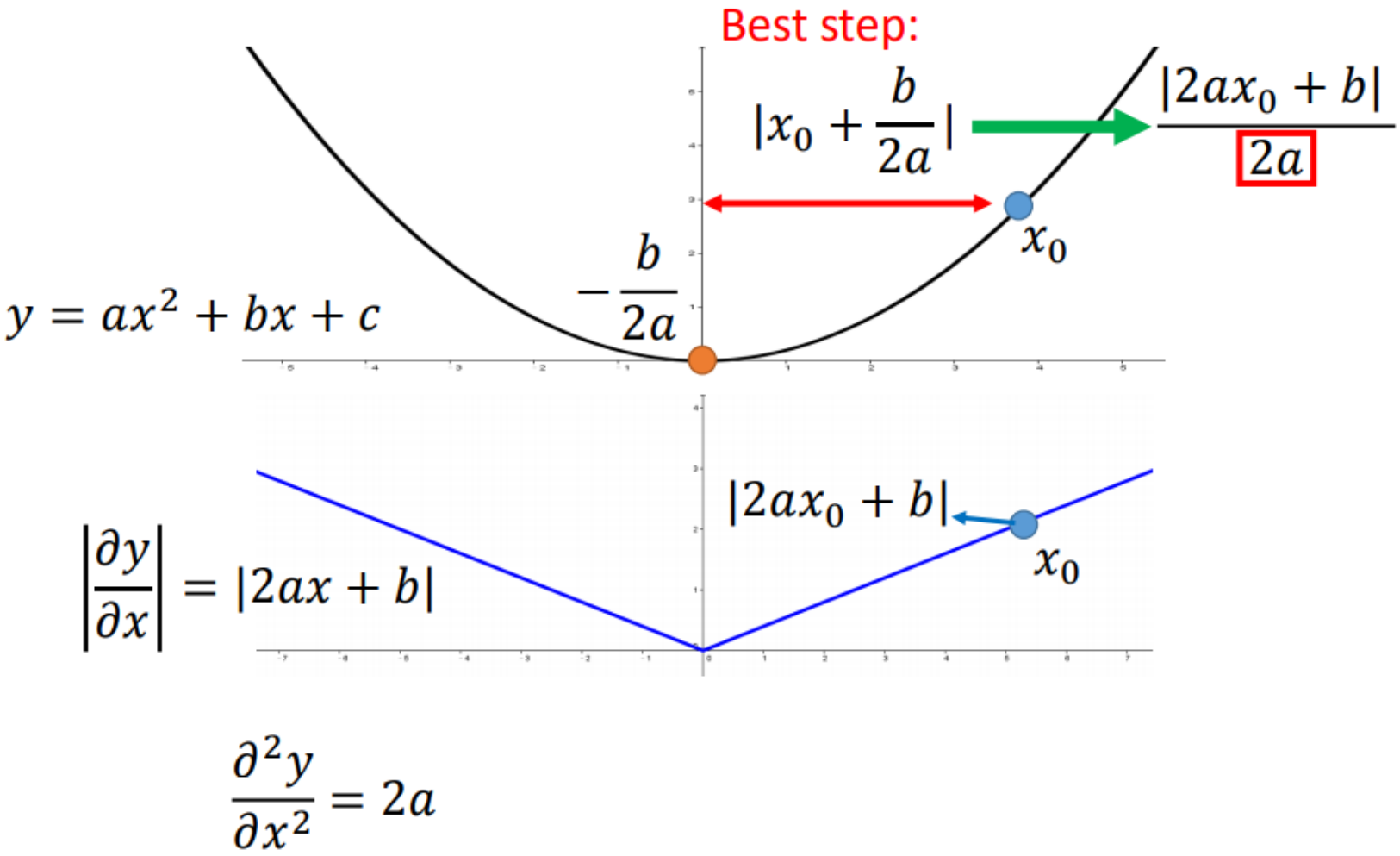


Consider More Than One Parameter

- How about a and c?
- The 1st order derivation of a is smaller than c. But a is farther from the minimum point.



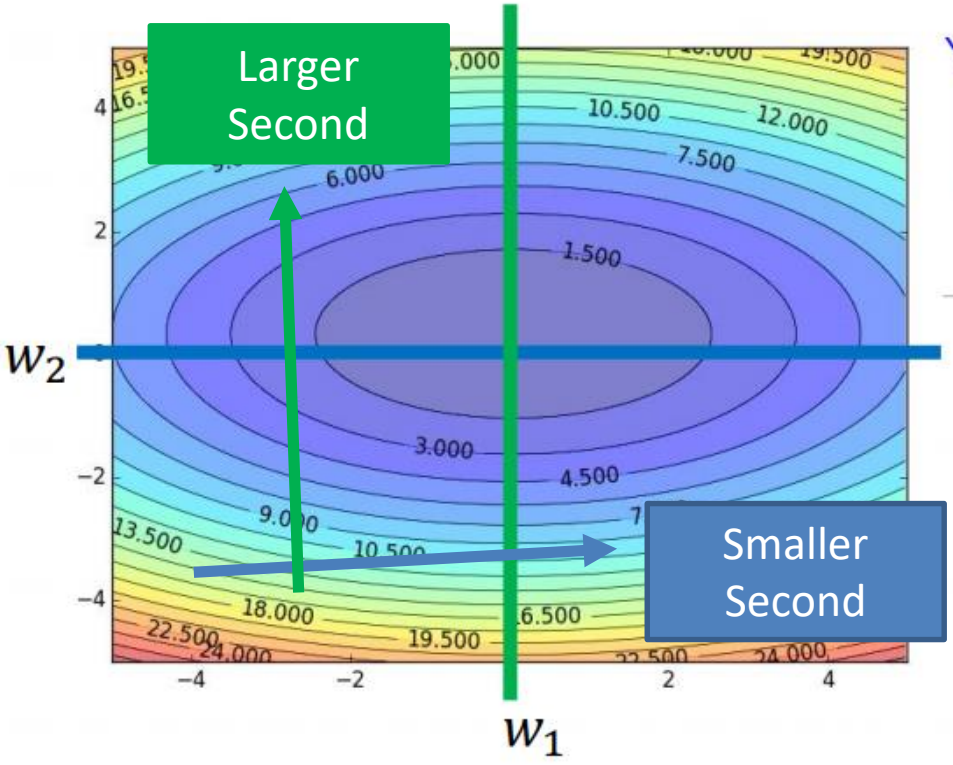
Second Derivative

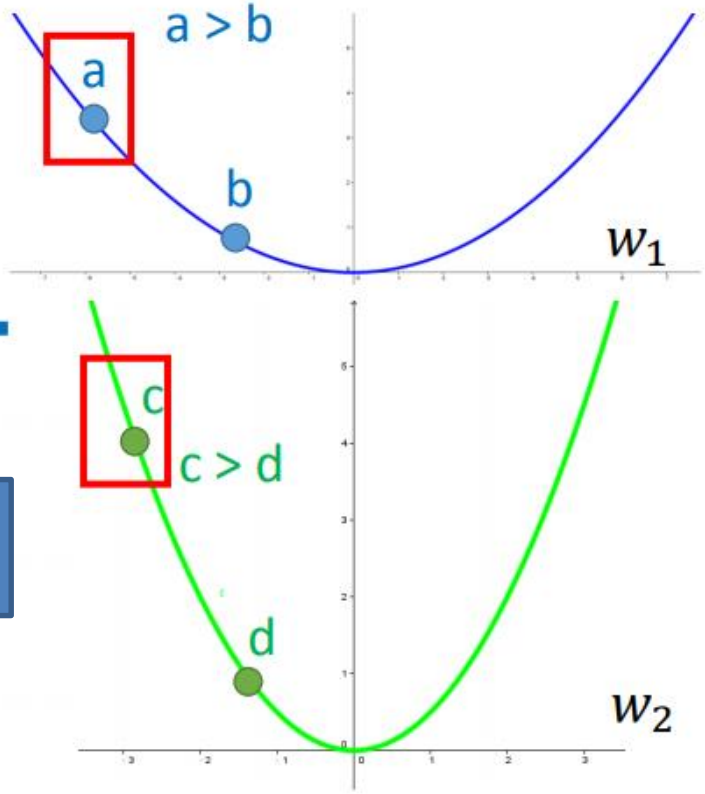


$$\frac{\partial^2 y}{\partial x^2} = 2a$$

Second Derivative Between Many Parameters

Smaller second derivative





The best step is $\frac{|\text{First derivative}|}{\text{Second derivative}}$

Larger second derivative

Second Derivative and Adagrad

- When data is larger, the computation of derivation needs a lot of time.
- Adopting first derivation result to estimate second derivative.

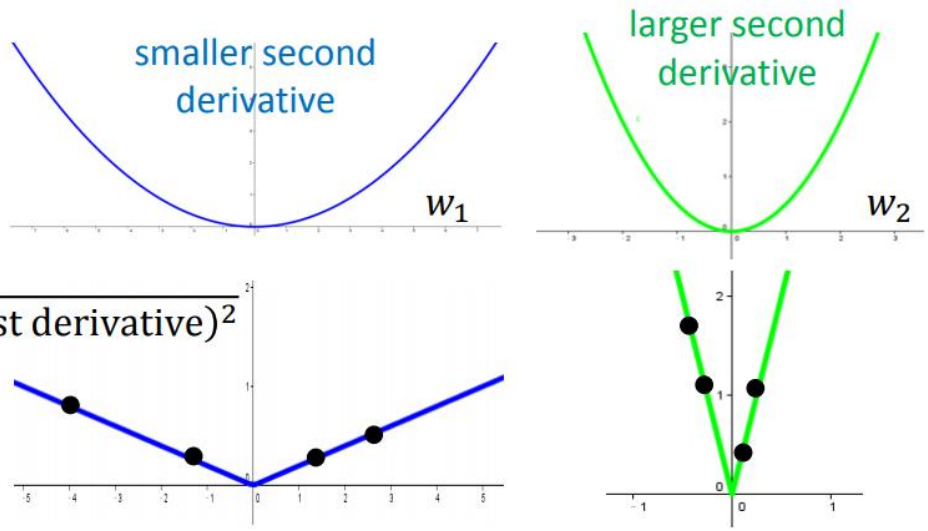
$$w^{t+1} \leftarrow w^t - \frac{\eta}{\sqrt{\sum_{i=0}^{t}(g^i)^2}} g^t$$

The best step is

$$\frac{|\text{First derivative}|}{\text{Second derivative}}$$

?

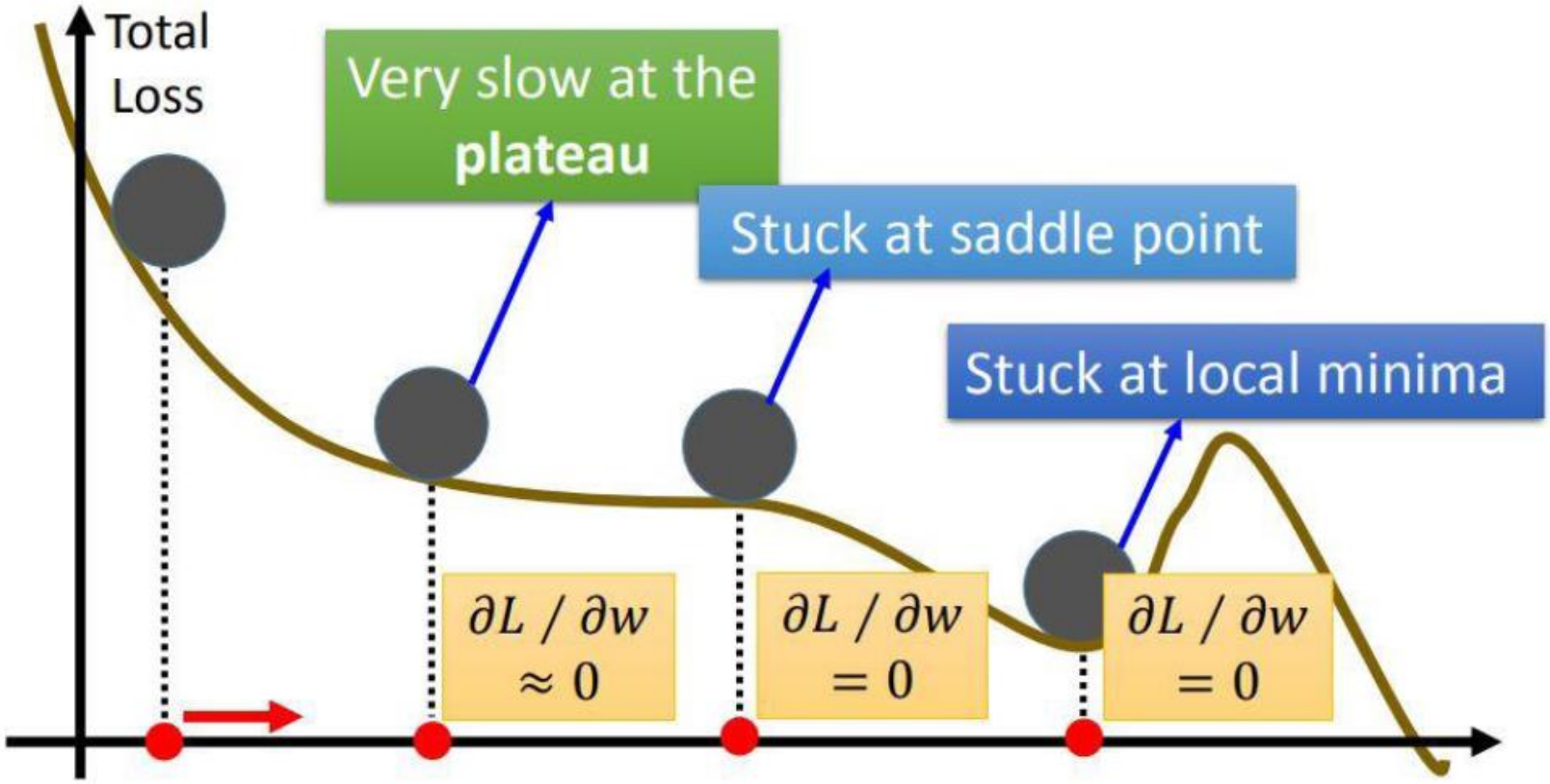
Use *first derivative* to estimate *second derivative*



Optimization

- Gradient decent
- Increase the performance of optimization
  - Tips 1:Tune Learning Rate
  - Tips2: Stochastic Gradient Descent
  - Tips3: Feature Scaling
- How to update many parameters

The Problem of Gradient Descent



Stochastic Gradient Descent

$$L = \sum_n \left(\hat{y}^n - \left(b + \sum w_i x_i^n \right) \right)^2$$

Loss is the summation over all training examples

◆ ***Gradient Descent*** $\theta^i = \theta^{i-1} - \eta \nabla L\left(\theta^{i-1}\right)$

◆ ***Stochastic Gradient Descent*** Faster!

Pick an example x^n

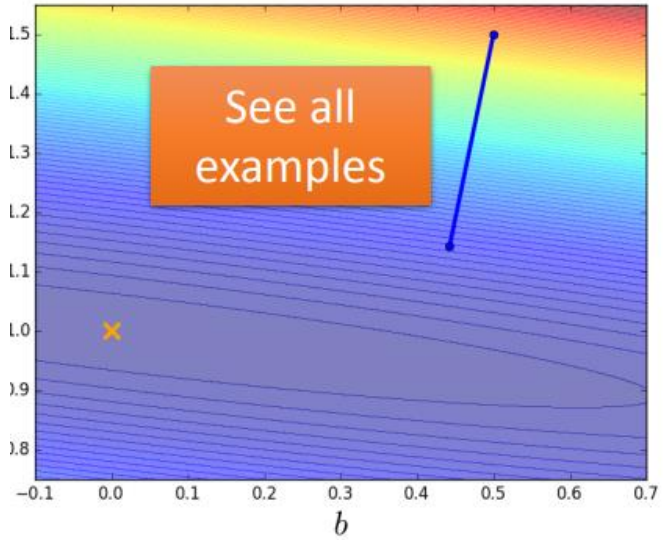
$$L^n = \left(\hat{y}^n - \left(b + \sum w_i x_i^n \right) \right)^2 \qquad \theta^i = \theta^{i-1} - \eta \nabla L^n\left(\theta^{i-1}\right)$$

Loss for only one example

Example

Gradient Descent

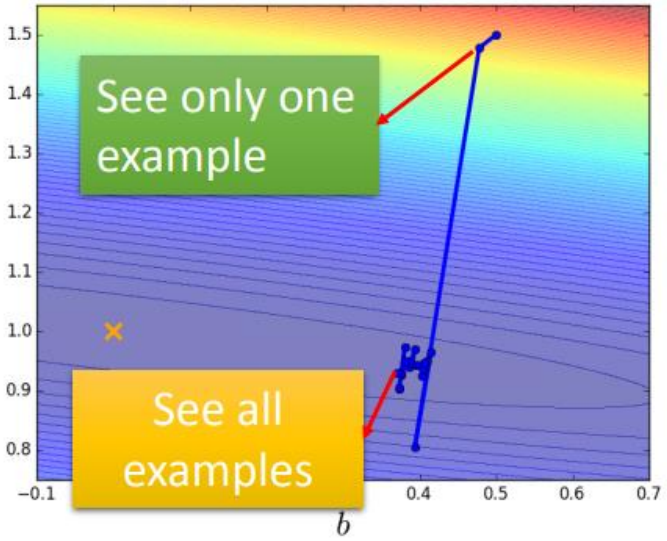
Update after seeing all examples



Stochastic Gradient Descent

Update for each example

If there are 20 examples, 20 times faster.



- To increase the performance of SGD, we can update by mini-batch SGD.

Stochastic Gradient Descent and Mini-batch

◆***Gradient Descent***

$$\theta^i = \theta^{i-1} - \eta \nabla L\left(\theta^{i-1}\right) \qquad \nabla C\left(\theta^{i-1}\right) = \frac{1}{R}\sum_r \nabla L^r\left(\theta^{i-1}\right)$$

◆***Stochastic Gradient Descent***

Pick an example x_r $\qquad \theta^i = \theta^{i-1} - \eta \nabla L^r\left(\theta^{i-1}\right)$

◆***Mini Batch Gradient Descent***

Pick B examples as a batch b

B is batch size

Shuffle your data

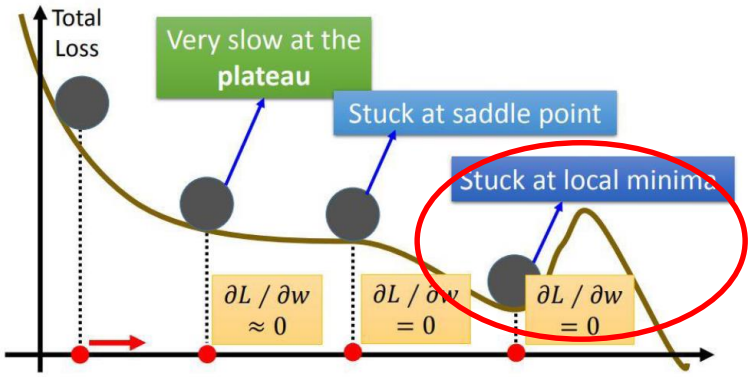
$$\theta^i = \theta^{i-1} - \eta \frac{1}{B}\sum_{x_r \in b} \nabla L^r\left(\theta^{i-1}\right)$$

Average the gradient of the examples in the batch b

Stochastic Gradient Decent with Momentum

◆Stochastic Gradient Descent

$$\theta^i = \theta^{i-1} - \eta \nabla L^r\left(\theta^{i-1}\right)$$



- Momentum means “volume.”
- Learning speed will be faster if the learning direction is same with moving direction.
- Otherwise, the learning speed is slower.

◆Stochastic Gradient Descent with Momentum

$$v^i = \lambda v^{i-1} - \eta \nabla L^r\left(\theta^{i-1}\right)$$

$$\theta^i = \theta^{i-1} + v^i$$

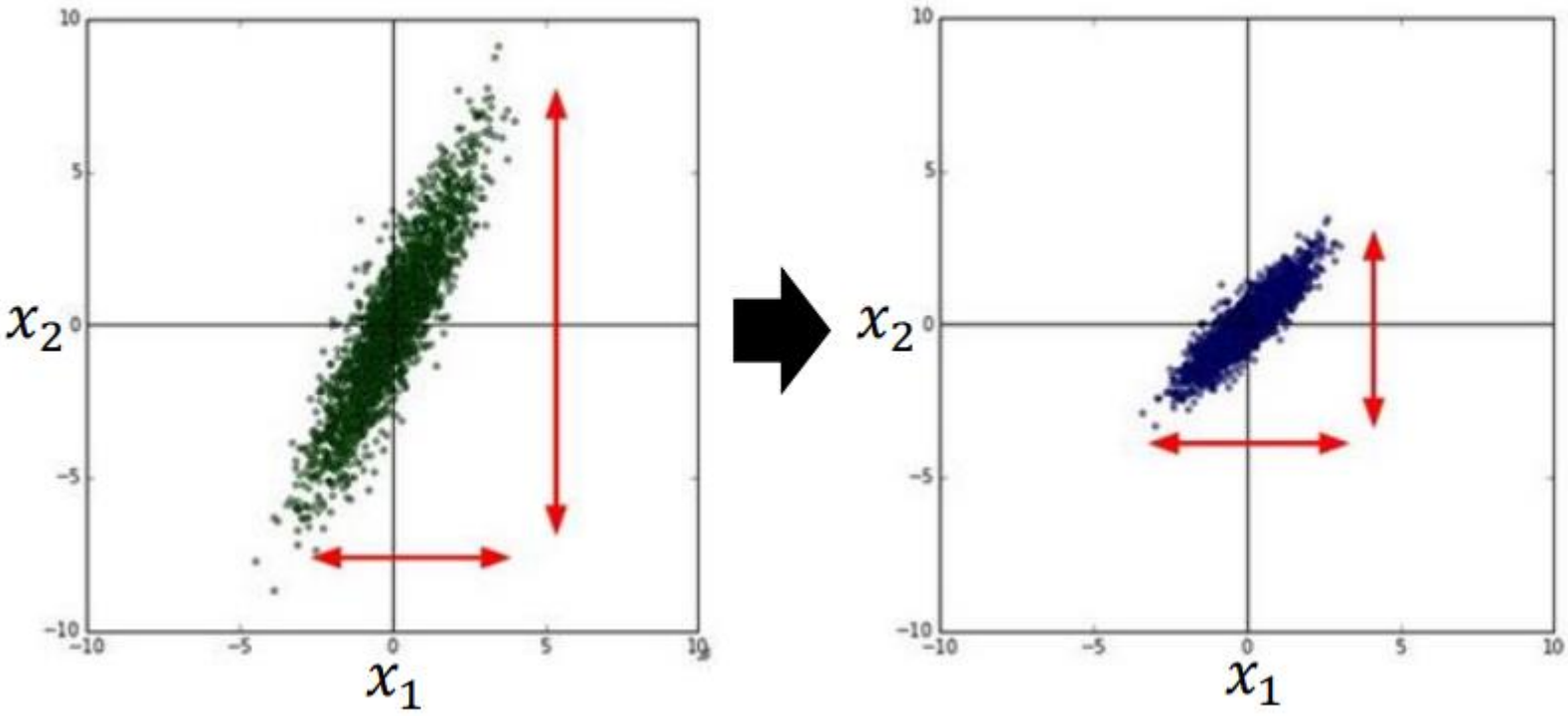
Optimization

- Gradient decent
- Increase the performance of optimization
  - Tips 1:Tune Learning Rate
  - Tips2: Stochastic Gradient Descent
  - Tips3: Feature Scaling
- How to update many parameters

Feature Scaling

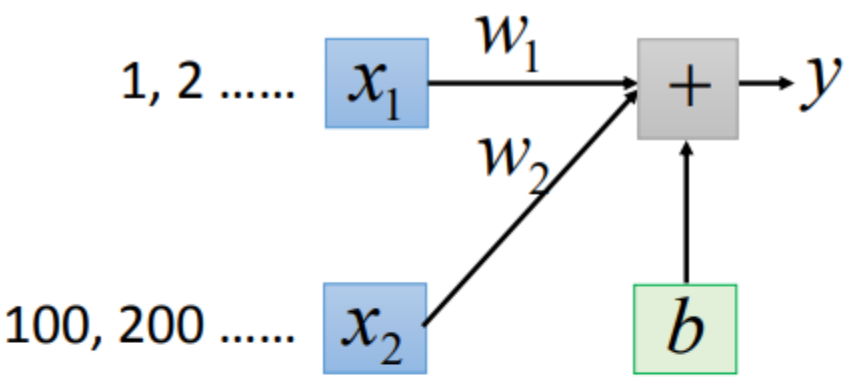
- Make the scale of different features have the same scaling.

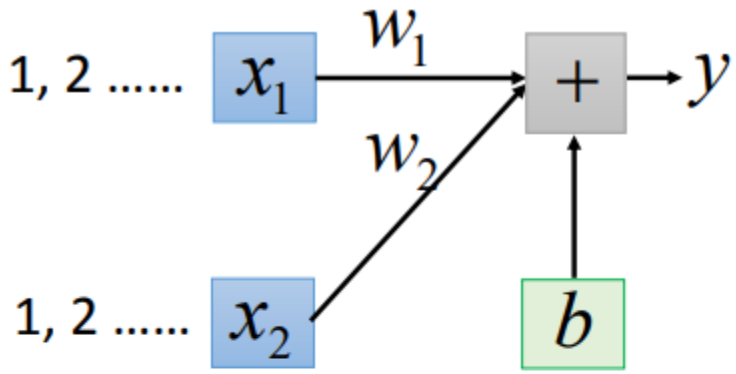
$$y = b + w_1 x_1 + w_2 x_2$$

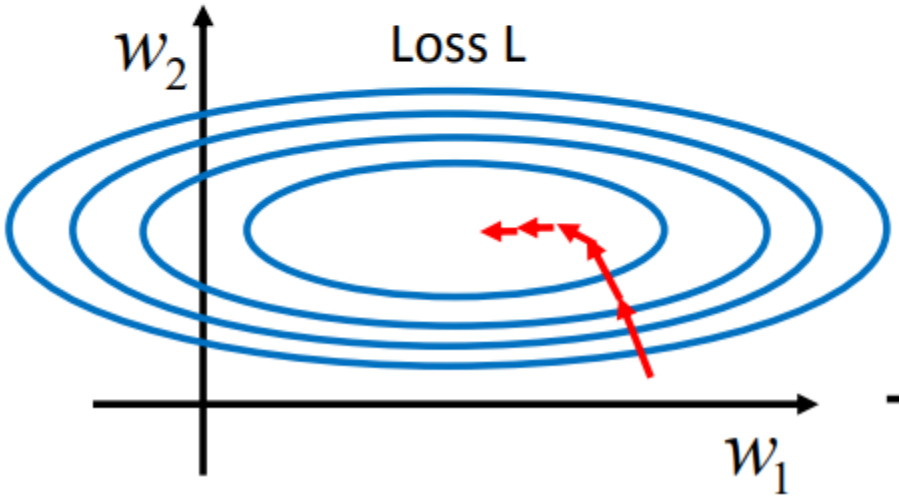


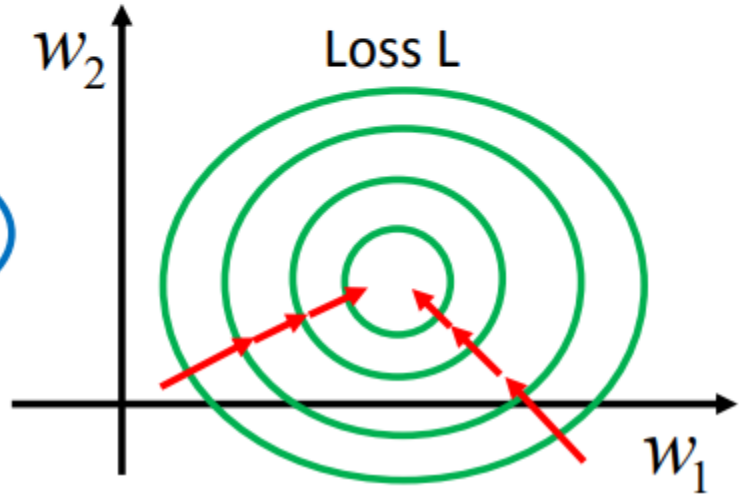
Feature Scaling

$$y = b + w_1 x_1 + w_2 x_2$$

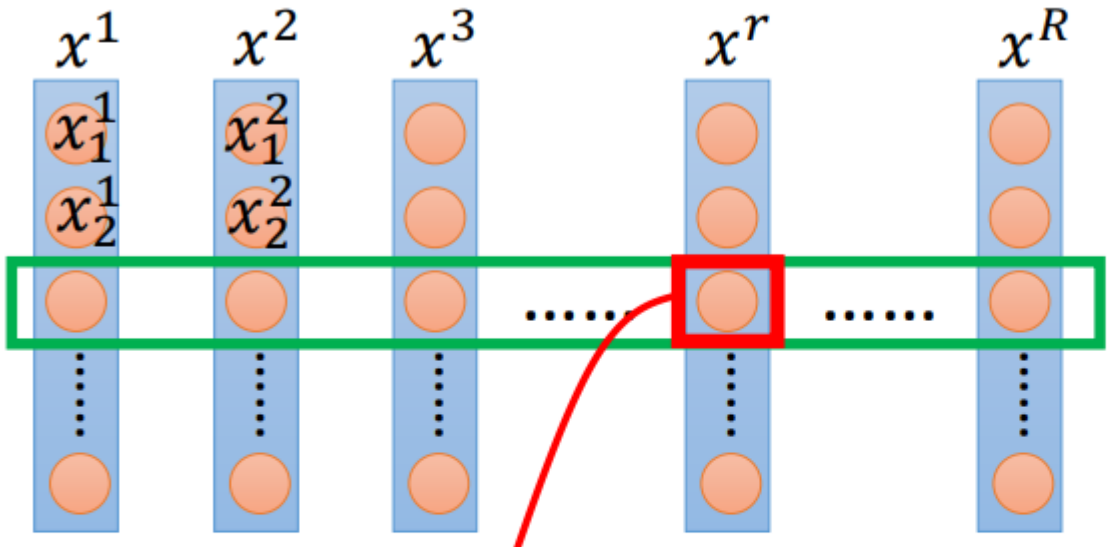








The Popular Method of Feature Scaling



For each dimension i:

mean: m_i

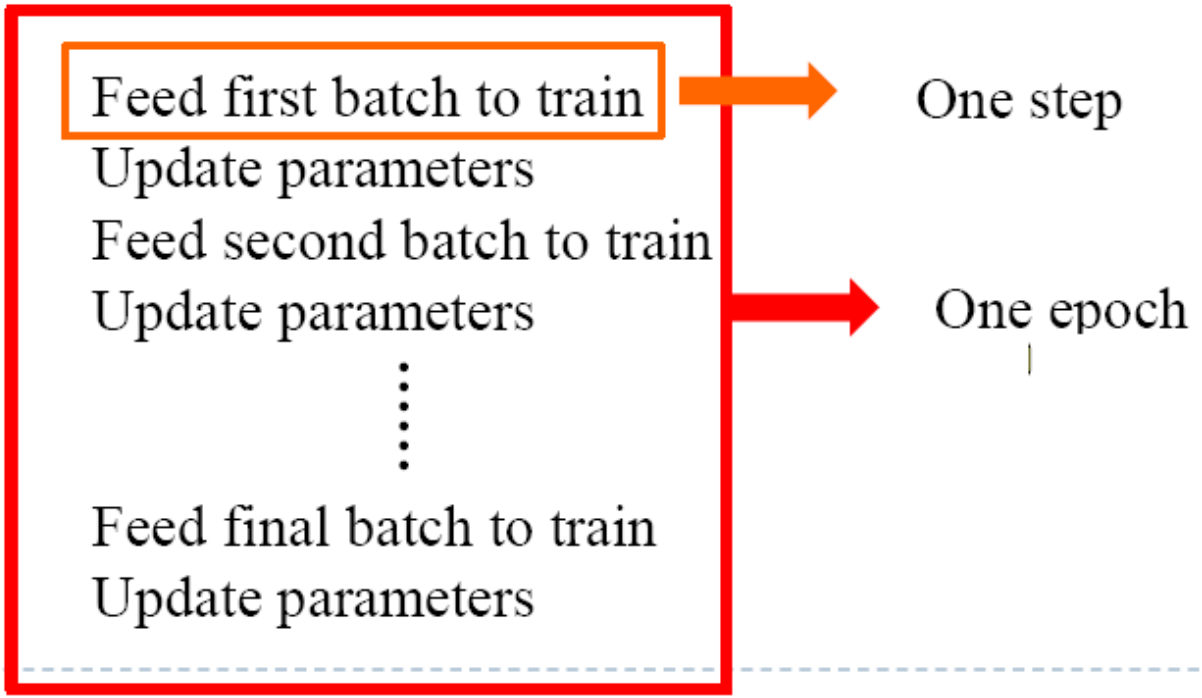
standard deviation: σ_i

$$x_i^r \leftarrow \frac{x_i^r - m_i}{\sigma_i}$$

The means of all dimensions become 0.
The variances are all 1.

What is one epoch/one step?

- One step
  - Pass a batch of training data (batch size is user define)
- One epoch
  - One pass of all training data



Optimization

- Gradient decent
- Increase the performance of optimization
  - Tips 1:Tune Learning Rate
  - Tips2: Stochastic Gradient Descent
  - Tips3: Feature Scaling
- How to update many parameters

Gradient Descent

Network parameters $\theta = \{w_1, w_2, \cdots, b_1, b_2, \cdots\}$

Starting Parameters $\theta^0 \longrightarrow \theta^1 \longrightarrow \theta^2 \longrightarrow \cdots\cdots$

$$\nabla L(\theta) = \begin{bmatrix} \partial L(\theta)/\partial w_1 \\ \partial L(\theta)/\partial w_2 \\ \vdots \\ \partial L(\theta)/\partial b_1 \\ \partial L(\theta)/\partial b_2 \\ \vdots \end{bmatrix}$$

$Compute\ \nabla L(\theta^0)$ $\quad \theta^1 = \theta^0 - \eta \nabla L(\theta^0)$

$Compute\ \nabla L(\theta^1)$ $\quad \theta^2 = \theta^1 - \eta \nabla L(\theta^1)$

Millions of parameters ……

Backpropagation

- 1986, Rumelhar and Hinton proposed backpropagation to solve the complex computation in neural network.
- Neural network becomes popular.

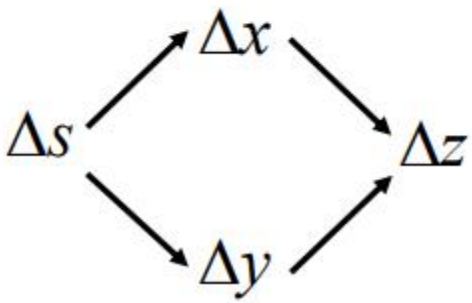
Recall: Chain Rule

Case 1 $y = g(x) \quad z = h(y)$

$$\Delta x \to \Delta y \to \Delta z \qquad \frac{dz}{dx} = \frac{dz}{dy}\frac{dy}{dx}$$

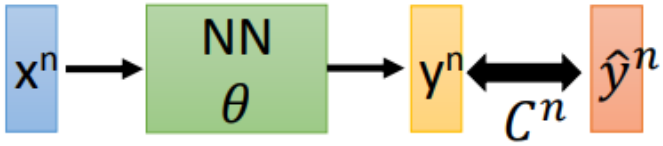
Case 2

$$x = g(s) \qquad y = h(s) \qquad z = k(x, y)$$

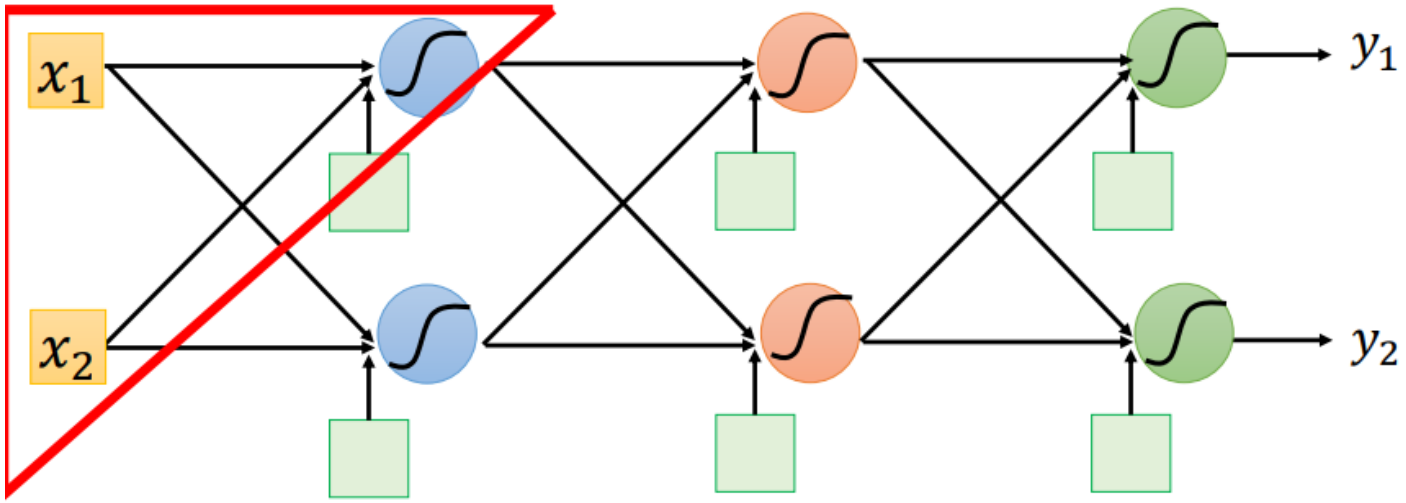


$$\frac{dz}{ds} = \frac{\partial z}{\partial x}\frac{dx}{ds} + \frac{\partial z}{\partial y}\frac{dy}{ds}$$

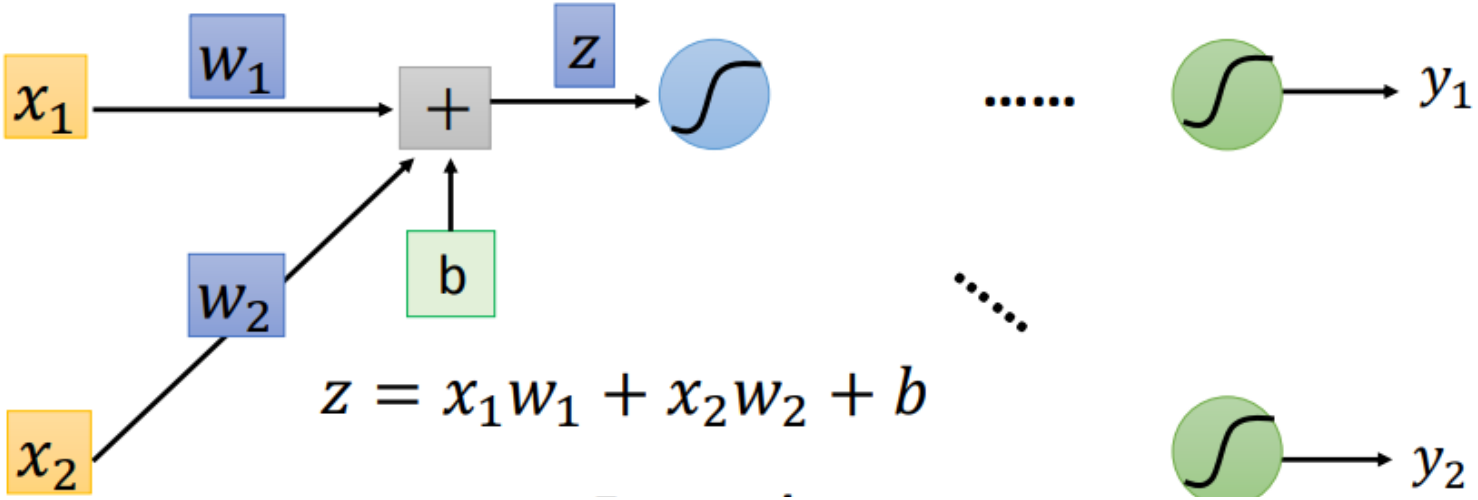
Backpropagation



$$L(\theta) = \sum_{n=1}^{N} C^n(\theta) \quad \Rightarrow \quad \frac{\partial L(\theta)}{\partial w} = \sum_{n=1}^{N} \frac{\partial C^n(\theta)}{\partial w}$$



Backpropagation



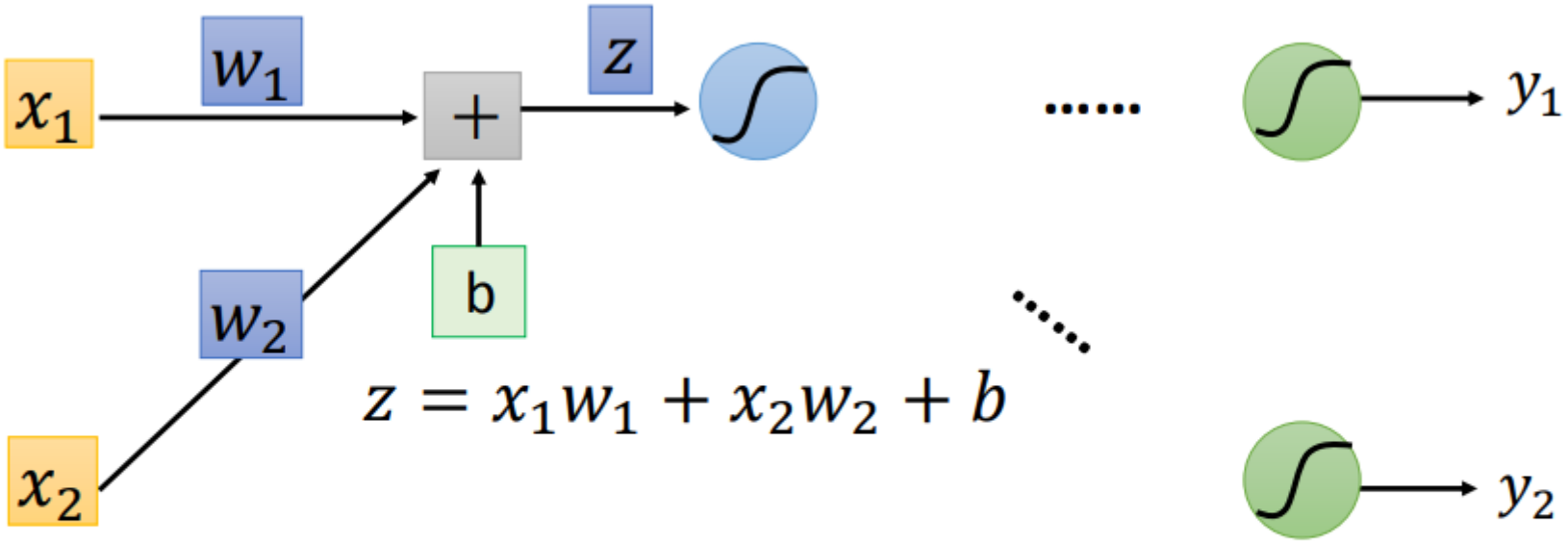
$$z = x_1 w_1 + x_2 w_2 + b$$

$$\frac{\partial C}{\partial w} = ? \quad \frac{\partial z}{\partial w}\frac{\partial C}{\partial z}$$

(Chain rule)

Backpropagation-Forward Pass

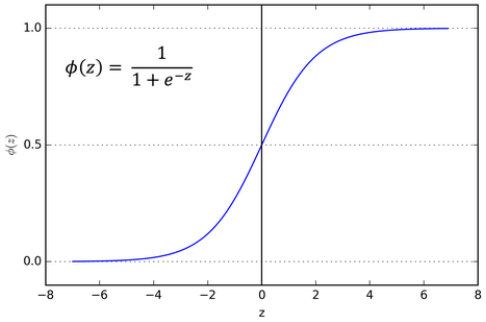
Compute $\partial z/\partial w$ for all parameters



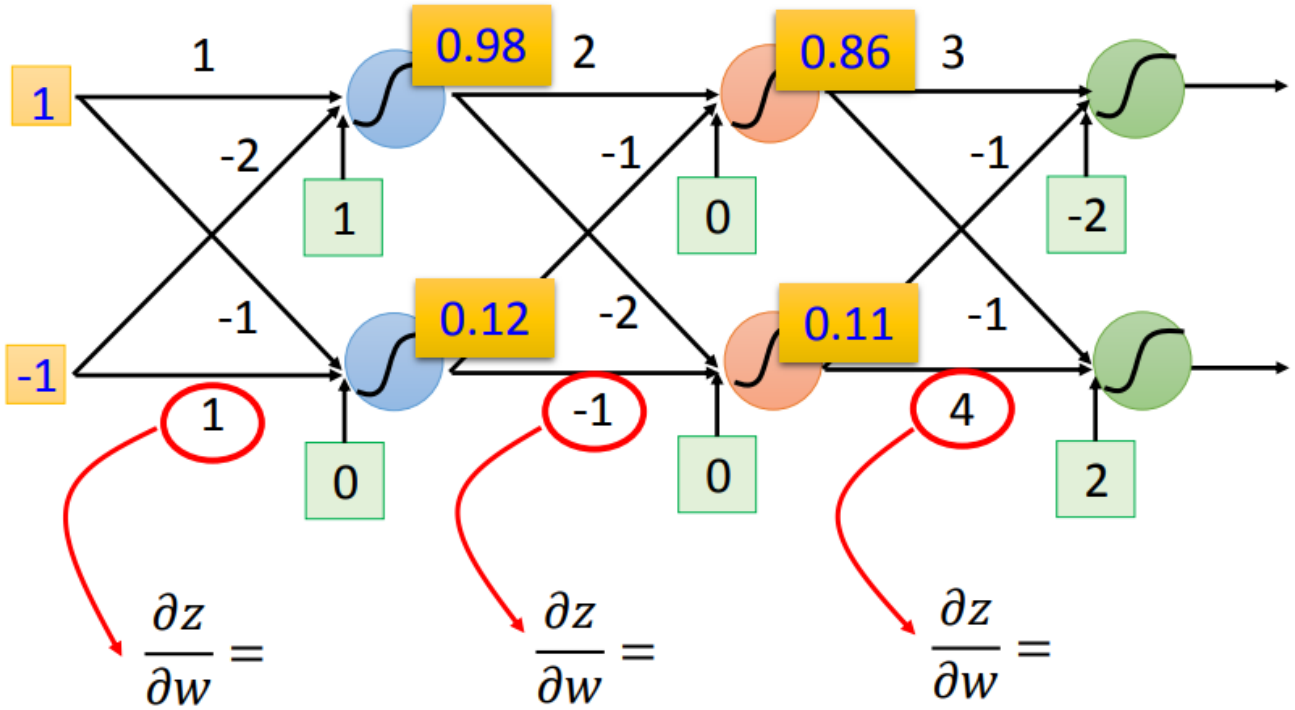
$$z = x_1 w_1 + x_2 w_2 + b$$

Example of Forward Pass

- Assume the activation function is sigmoid.



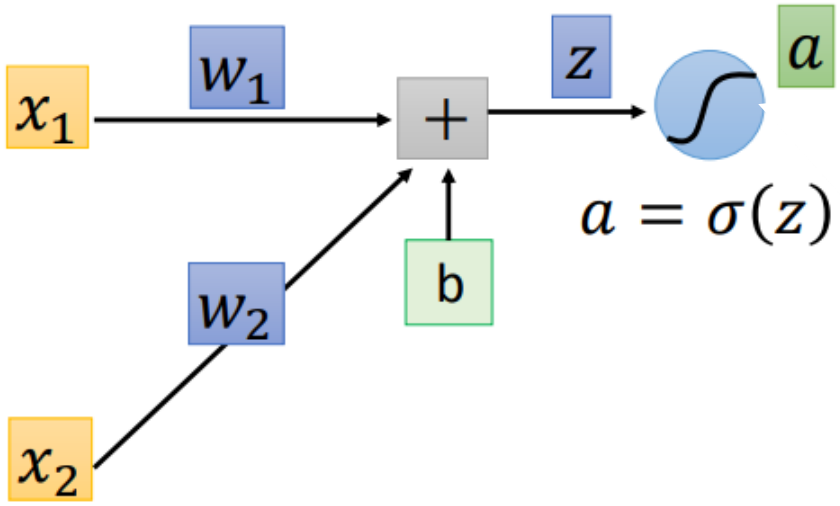
Compute $\partial z/\partial w$ for all parameters



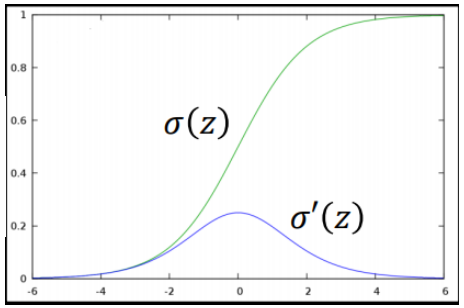
Backpropagation-Backward Pass

- Assume the activation function is sigmoid. $a = \sigma(z)$

Compute $\partial C / \partial z$ for all activation function inputs z

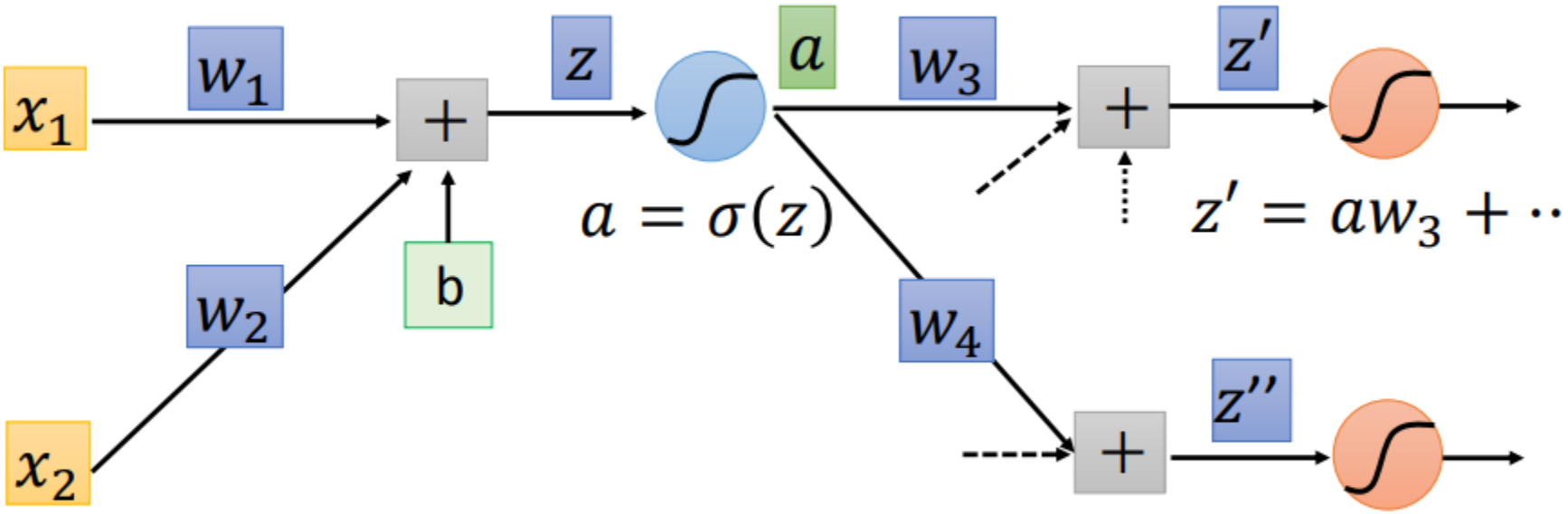


$\sigma'(z)$



Backpropagation-Backward Pass

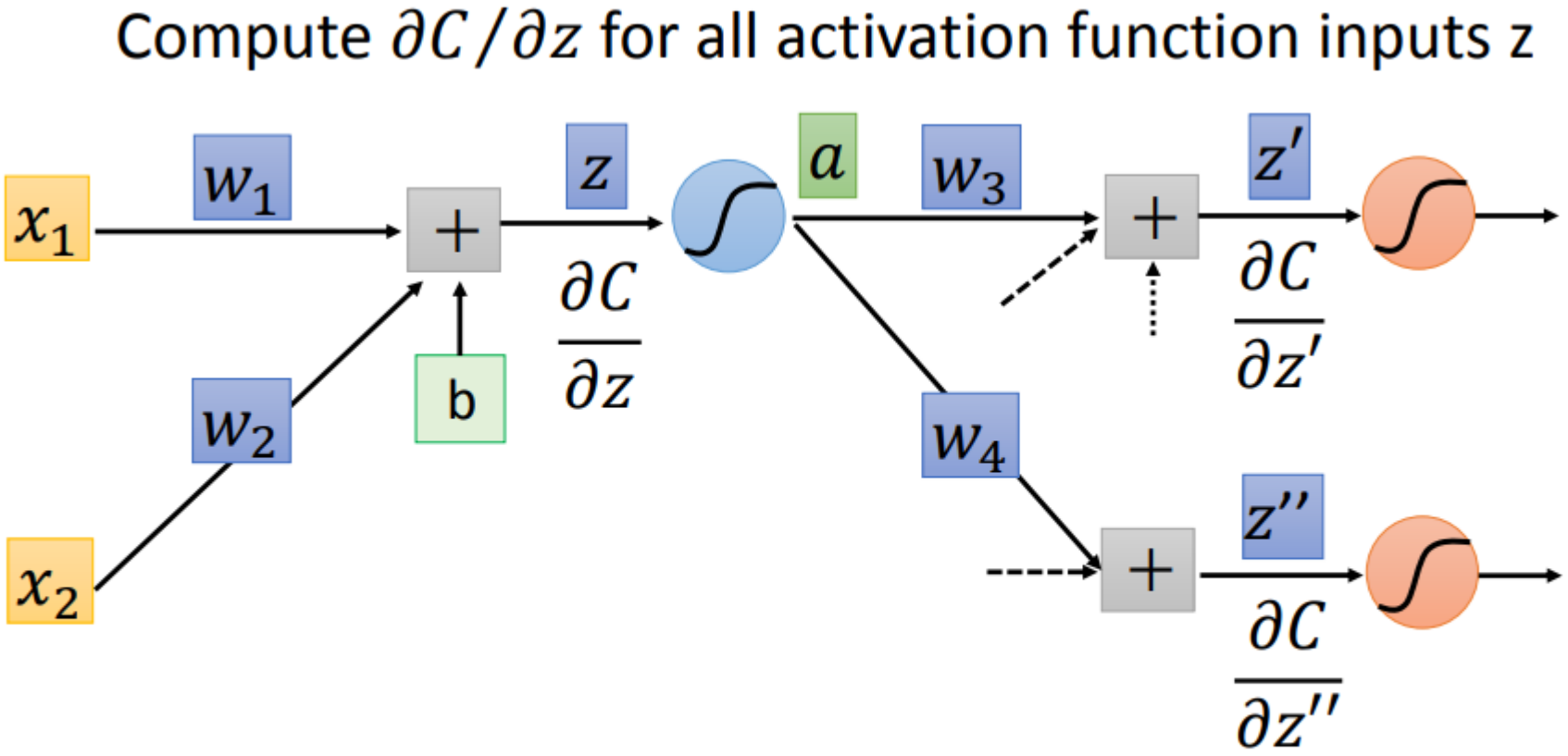
Compute $\partial C/\partial z$ for all activation function inputs z



$\sigma'(z)$ w_3 ? w_4 ?

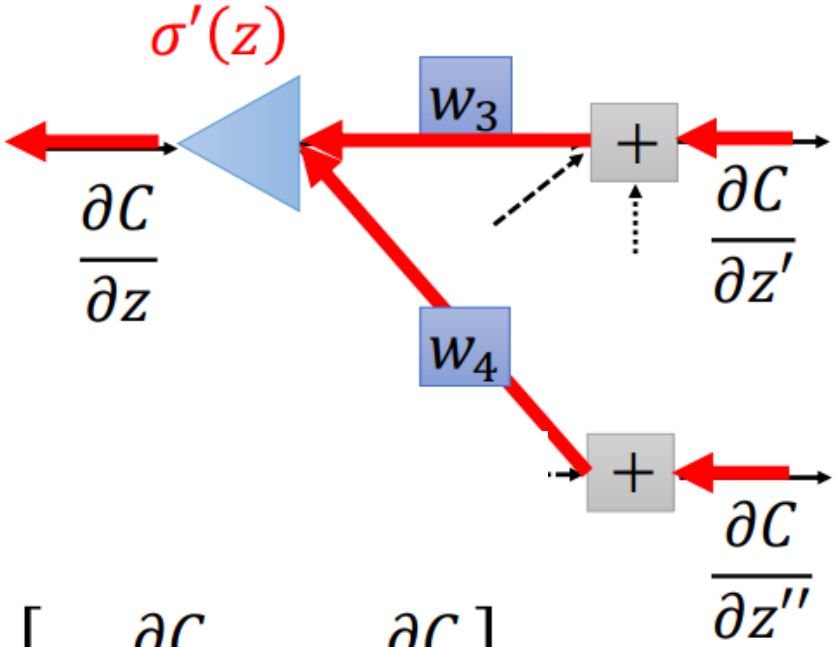
Assumed it's known

Backpropagation-Backward Pass



Backpropagation-Backward Pass

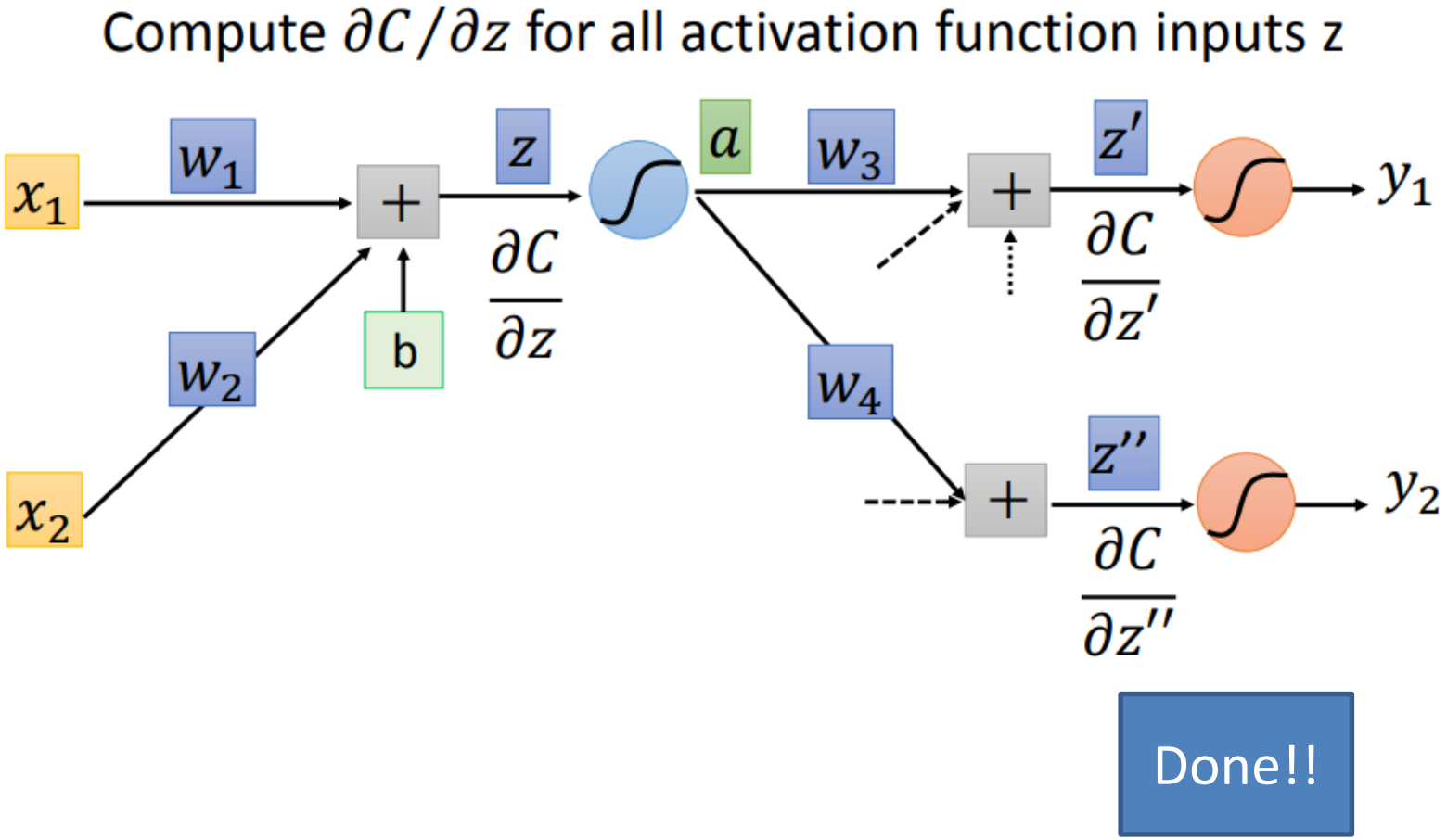
- Imagine there is another neuron.
- Input is $\frac{\partial C}{\partial z'}$ and $\frac{\partial C}{\partial z''}$



$$\frac{\partial C}{\partial z} = \sigma'(z)\left[w_3\frac{\partial C}{\partial z'} + w_4\frac{\partial C}{\partial z''}\right]$$

Backpropagation-Backward Pass

- Assume the orange neuron is output layer.

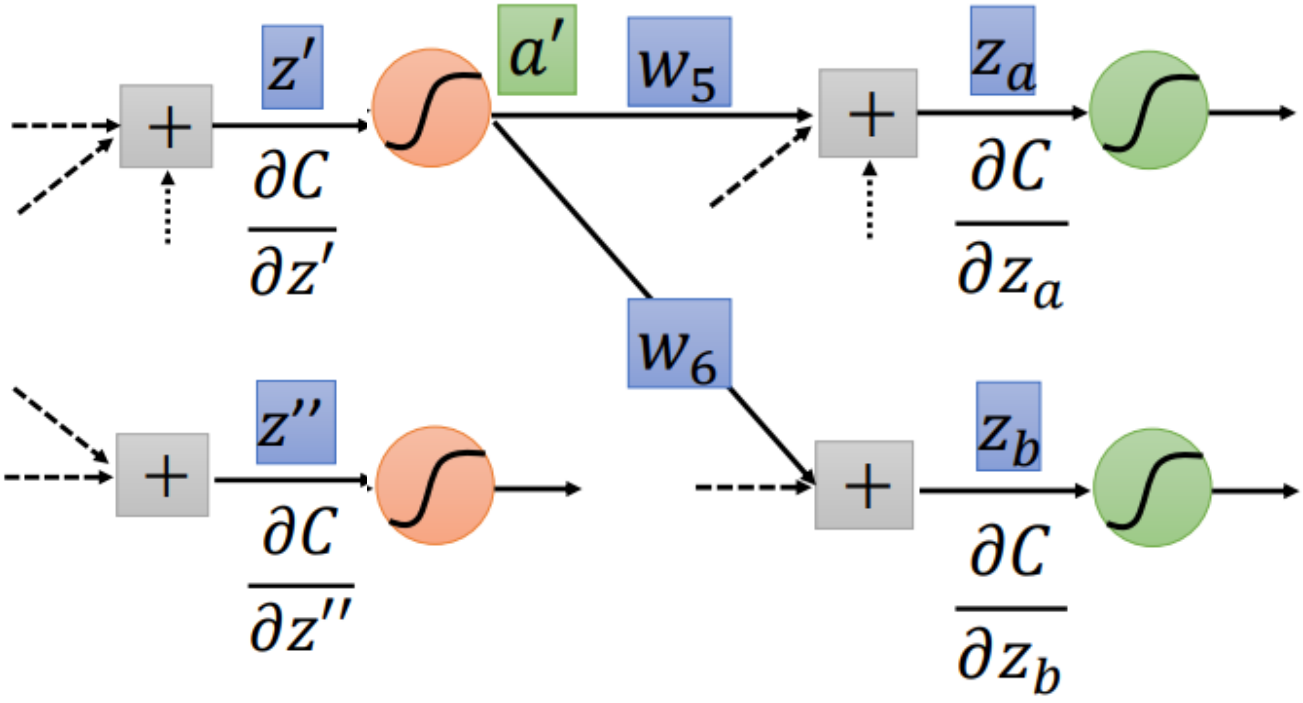


Backpropagation-Backward Pass

- If the orange neuron is not output layer

  Compute $\partial C/\partial z$ for all activation function inputs z

Case 2. Not Output Layer

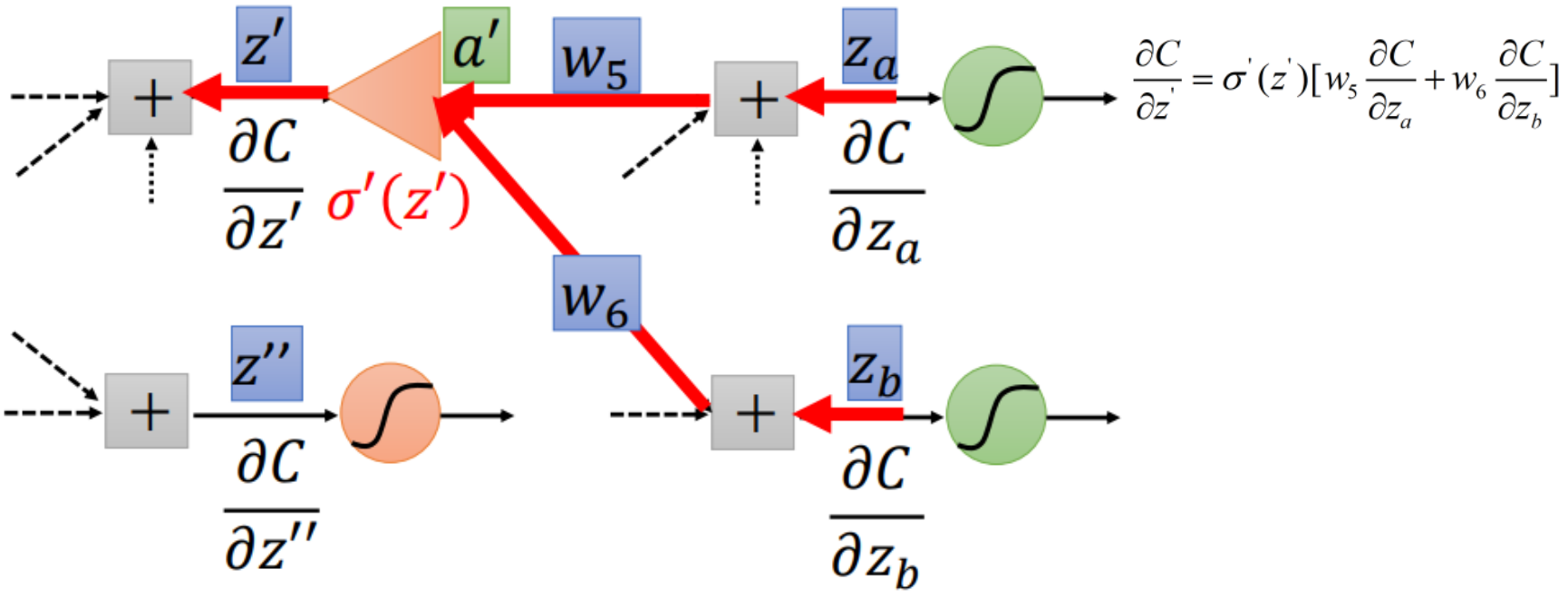


Backpropagation-Backward Pass

- If the orange neuron is not output layer

Compute $\partial C / \partial z$ for all activation function inputs z

Case 2. Not Output Layer

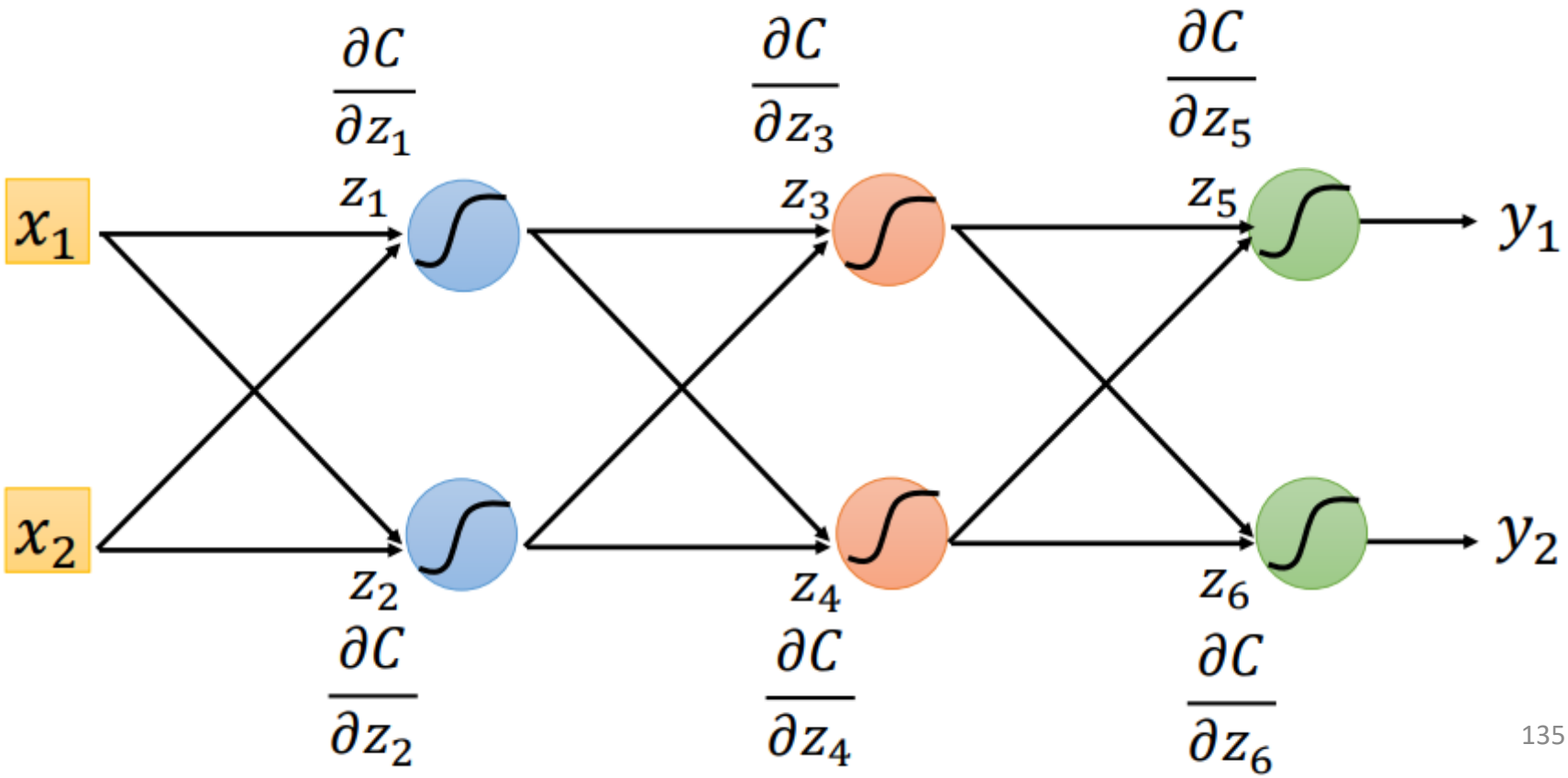


Backpropagation-Backward Pass

- Change direction

  Compute $\partial C/\partial z$ for all activation function inputs z

  Compute $\partial C/\partial z$ from the output layer



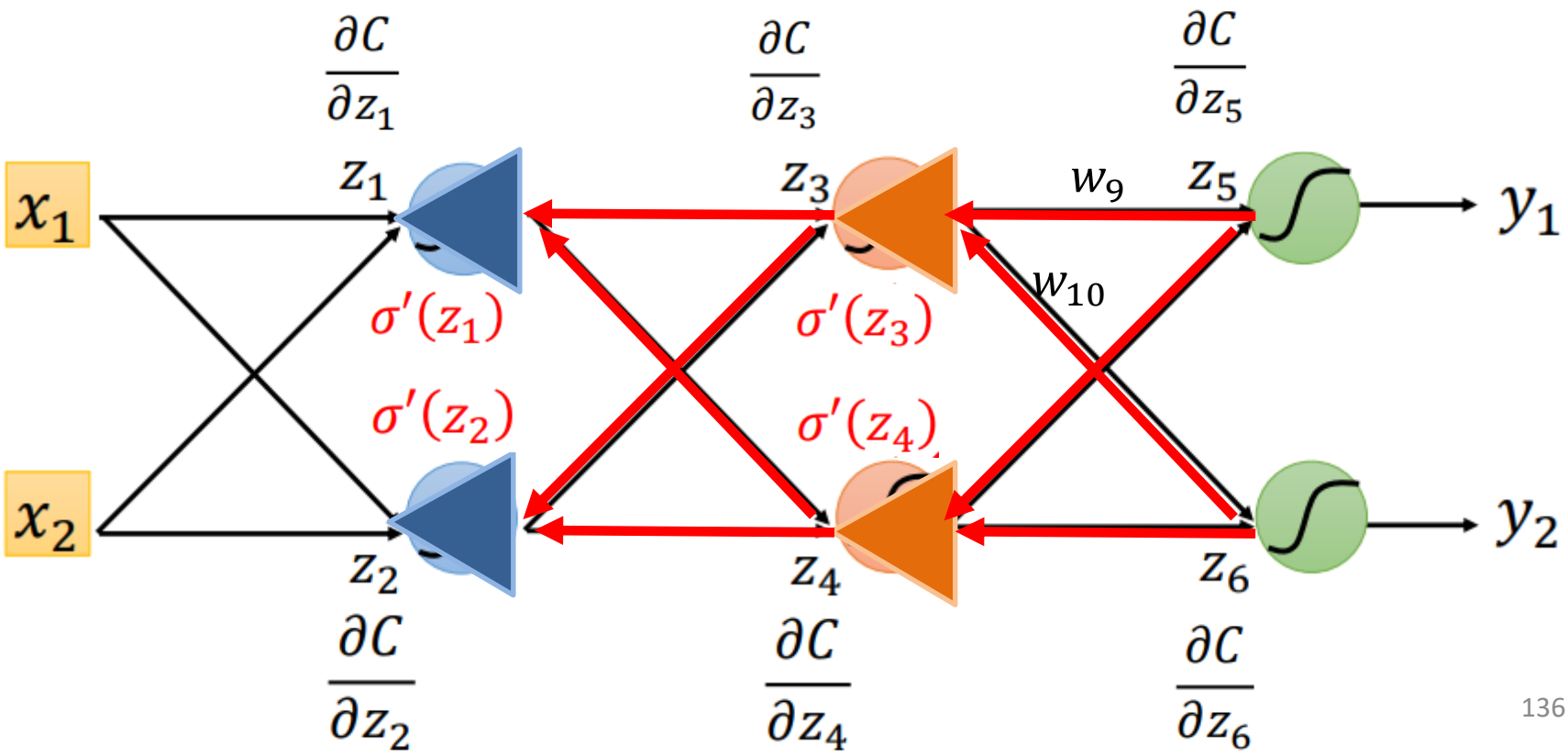
Backpropagation-Backward Pass

- Change direction

  Compute $\partial C/\partial z$ for all activation function inputs z

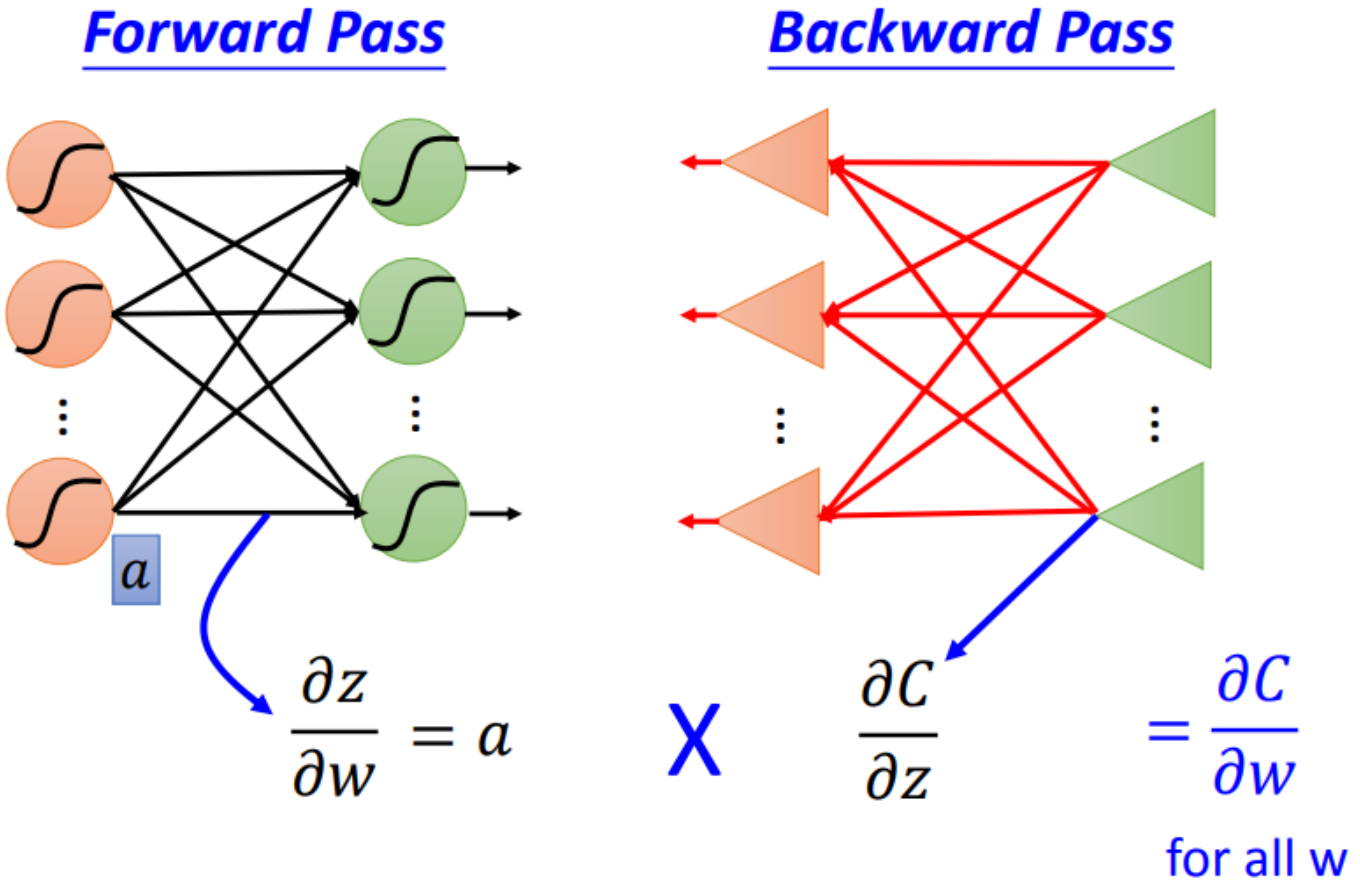
  Compute $\partial C/\partial z$ from the output layer

$$\frac{\partial C}{\partial z_3} = \sigma'(z_3)\left[w_9 \frac{\partial C}{\partial z_5} + w_{10} \frac{\partial C}{\partial z_6}\right]$$



Backpropagation-Summary

- The concept is constructing a opposite direction network.



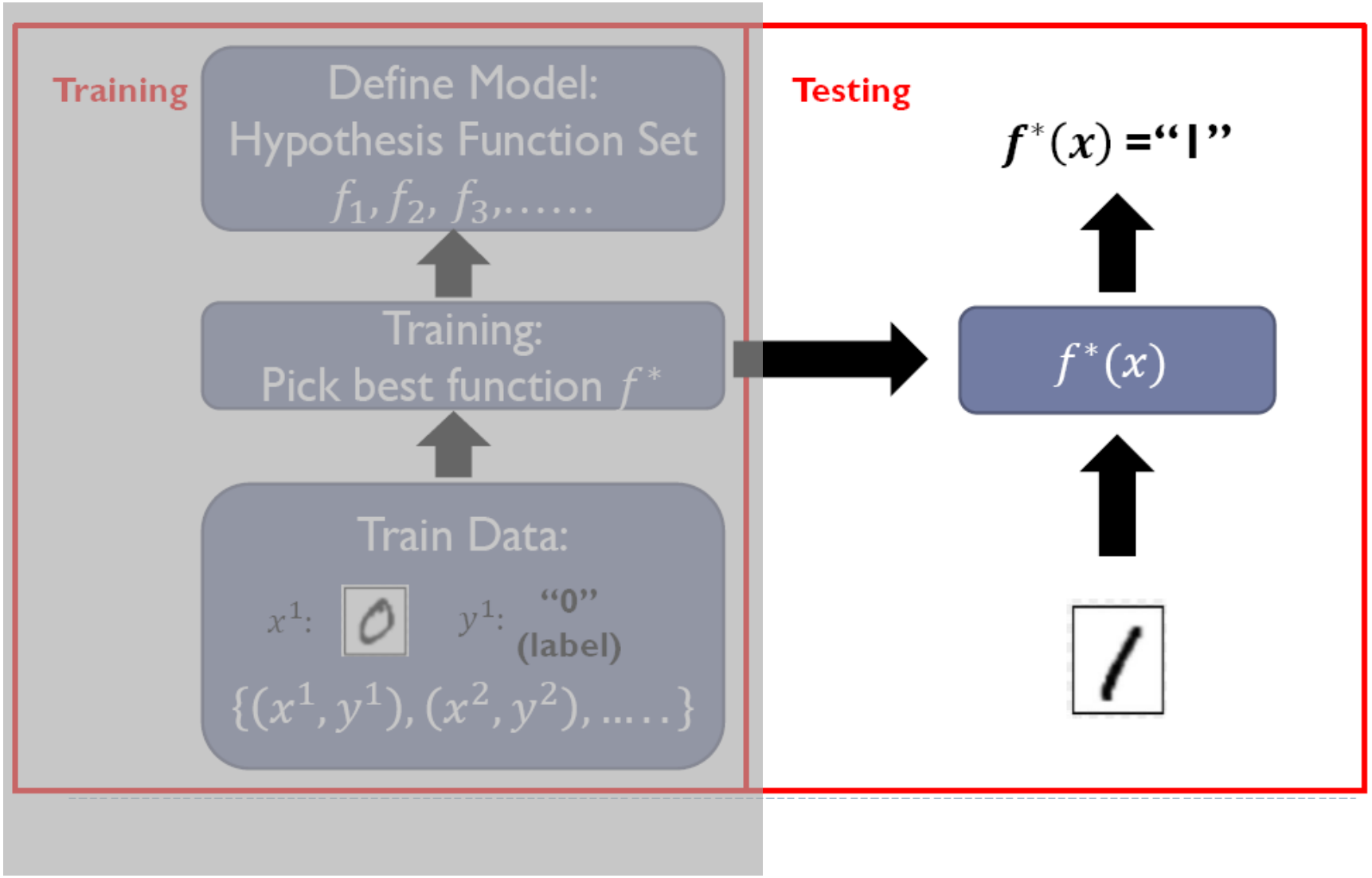
Backpropagation

- A network can have millions of parameters.
  - Backpropagation is the way to compute the gradients efficiently (not today)

- Many toolkits can compute the gradients automatically

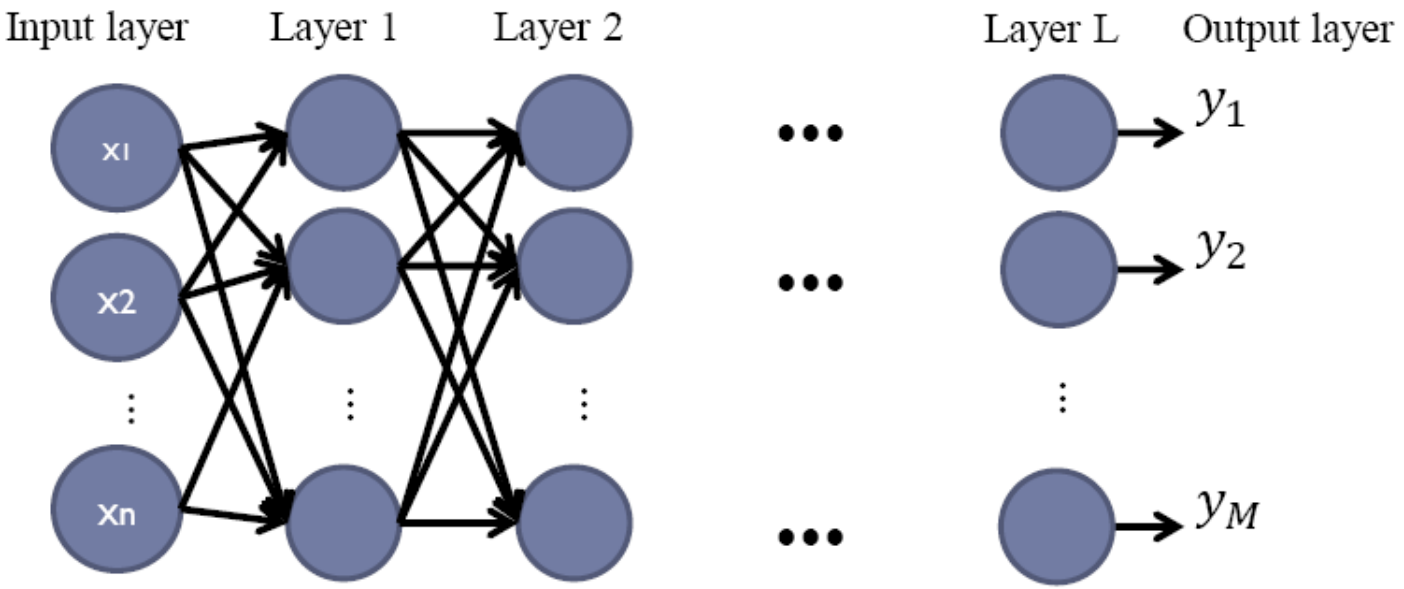
Testing Model

Learning


Training
Define Model:
Hypothesis Function Set
$f_1, f_2, f_3, \cdots\cdots$
Training:
Pick best function f^*
Train Data:
x^1:
y^1: "0" (label)
$\{(x^1, y^1), (x^2, y^2), \cdots\cdots\}$
Testing
$f^*(x)$ ="1"
$f^*(x)$

Testing: Predict/Validate Result

- "*" means a better set of parameters we have found.
- Use test data to find model accuracy.
  - # correct predict/# total data



$$\theta = \{W^{1*}, b^{1*}, W^{2*}, b^{2*}, \ldots, W^{L*}, b^{L*}\}$$

DNN Discussion

Outline

- Training Model
  - Prepare dataset
  - Build model
  - Define loss
    - Mean Square
    - Cross-entropy
  - Optimization
    - Gradient decent
    - Increase the performance of optimization
      - Tips 1:Tune Learning Rate
      - Tips2: Stochastic Gradient Descent
      - Tips3: Feature Scaling
    - How to update many parameters
- Testing Model
- Discussion

Discussion

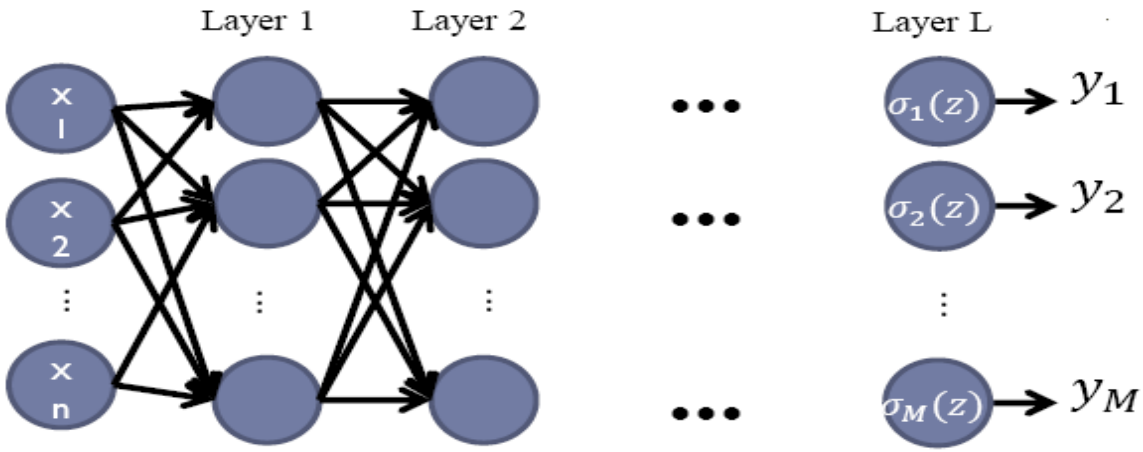
- Discussion1: Why DNN Fail in early day?
- Discussion2: Overfitting

Why DNN Fail in Early Day

- Artificial Neural Network was proposed in 1980.
- Some issued that mafe DNN fail in early day.
  - Hardware issue
    - Doesn't have fast enough GPU like today
  - Data issue
    - Hard to collect large amount of data
  - Activation Function
    - Use sigmoid as activation function

Activation Function

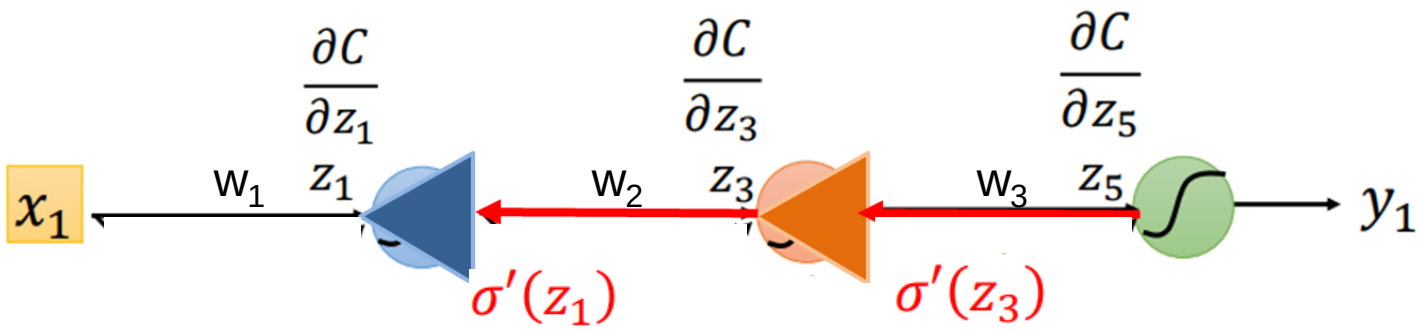
- Give NN nonlinear property
- Can be regarded as “ON(1) or OFF(0)”



$$\theta = \{W^1, b^1, W^2, b^2, \ldots, W^L, b^L\}$$

Backpropagation

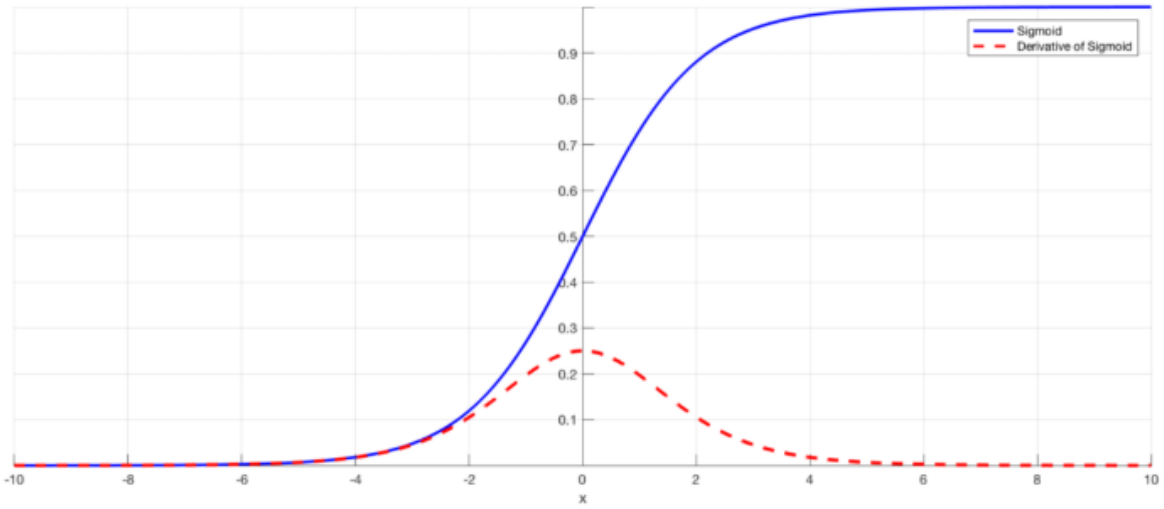
- Backpropagation = many multiplication of derivative activation function



$$\frac{\partial C}{\partial z_1} = \sigma'(z_1) w_2 \sigma'(z_3) w_3 \sigma'(z_5) \frac{\partial C}{\partial z_5}$$

Sigmoid Function

- The maximum value of derivative of sigmoid is 0.25.



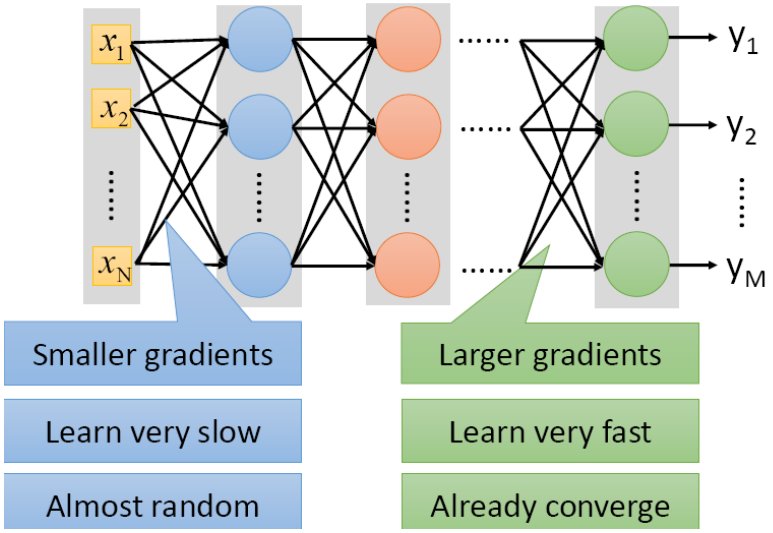
$$f(x)= \frac{1}{1+e^{-x}}$$

Vanishing Gradient Problem (1/2)

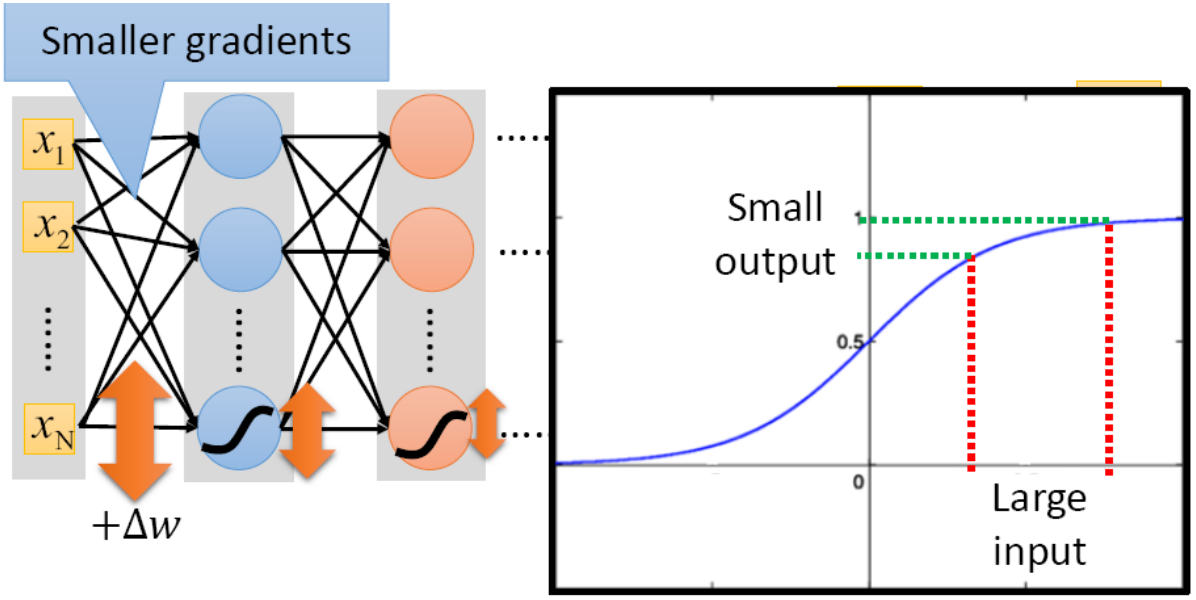
- The gradient of front layer is smaller than the back layer.
- Updating slow and slow……
- **=>Vanishing Gradient Problem.**

<0.25 <0.25

$$\frac{\partial C}{\partial z_1} = \sigma'(z_1) w_2 \sigma'(z_3) w_3 \sigma'(z_5) \frac{\partial C}{\partial z_5}$$

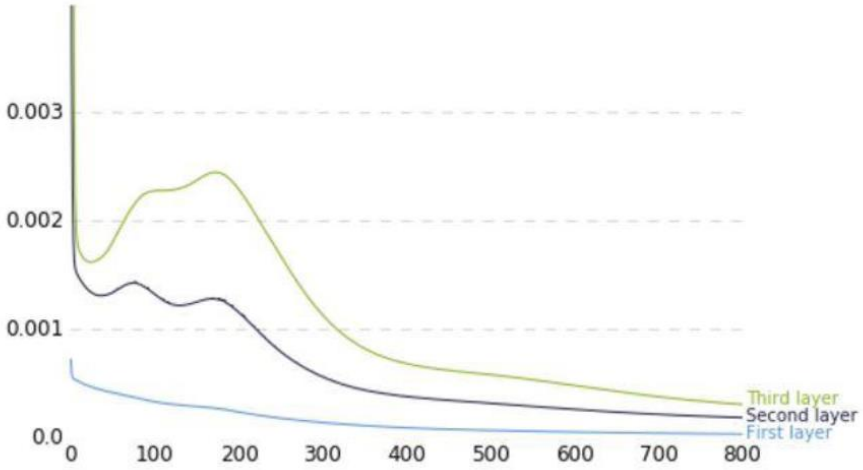


Vanishing Gradient Problem (2/2)



Intuitive way to compute the derivatives …

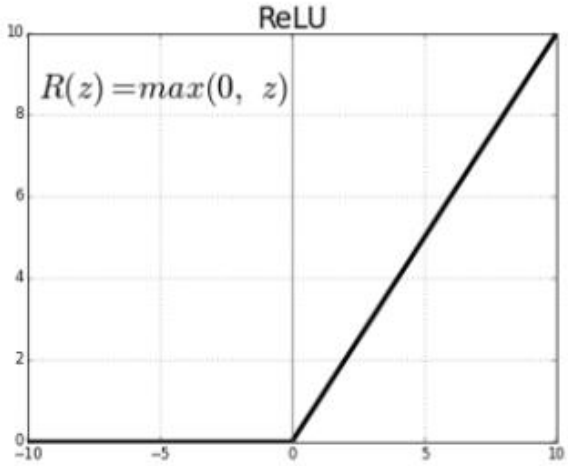
$$\frac{\partial l}{\partial w} =? \frac{\Delta l}{\Delta w}$$

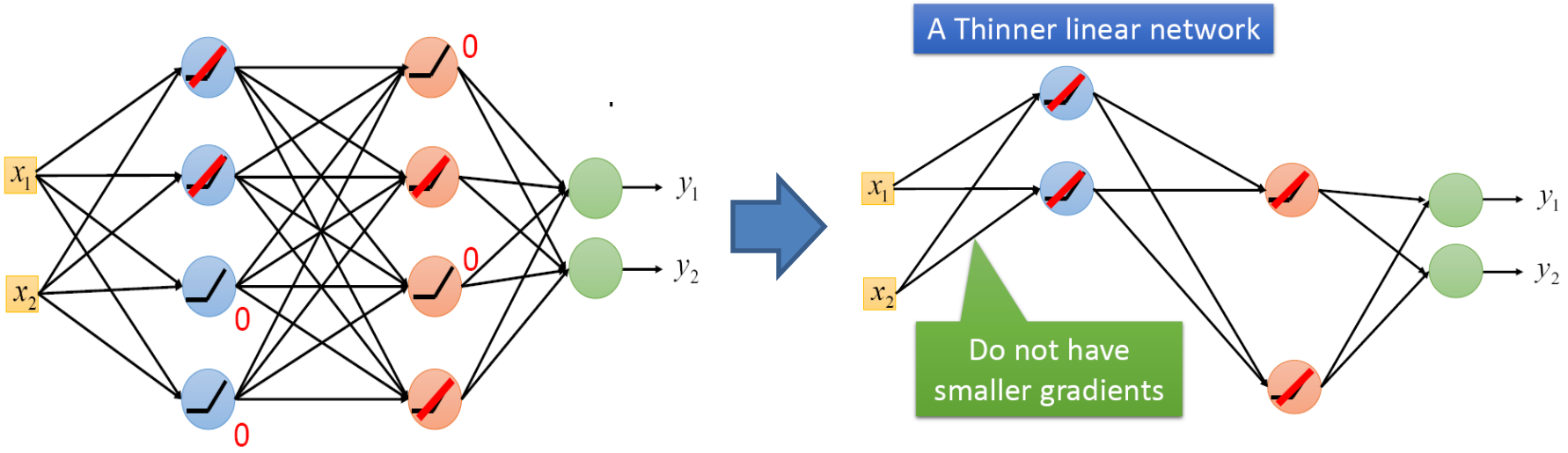


Solve Gradient Vanishing Problem (1/2)

- Change activation function: **ReLU**

$$y = \begin{cases} x, & if\ x \geq 0 \\ 0, & if\ x < 0 \end{cases}$$

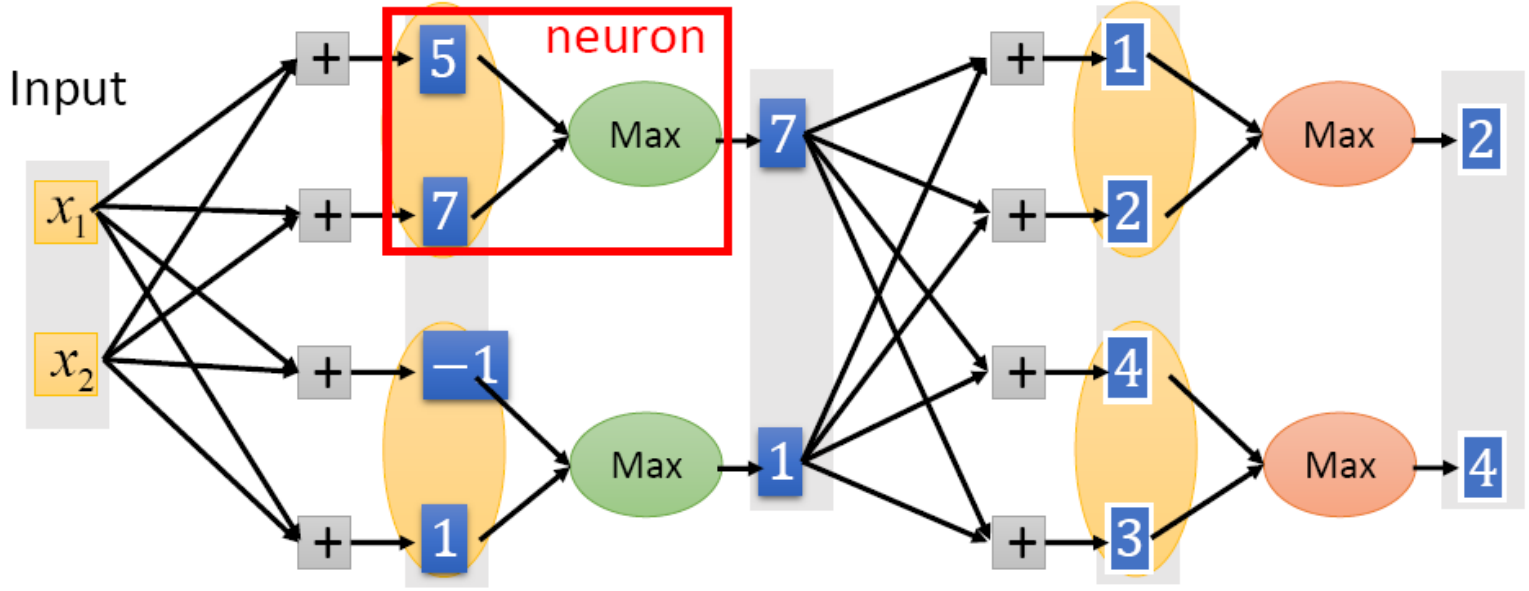




Delete unexpected neuron.

Solve Gradient Vanishing Problem (2/2)

- Change activation function: **Maxout**
- Learnable activation function



You can have more than 2 elements in a group.

ReLU is the special case of maxout.

Discussion

- Discussion1: Why DNN Fail in early day?
- Discussion2: Overfitting

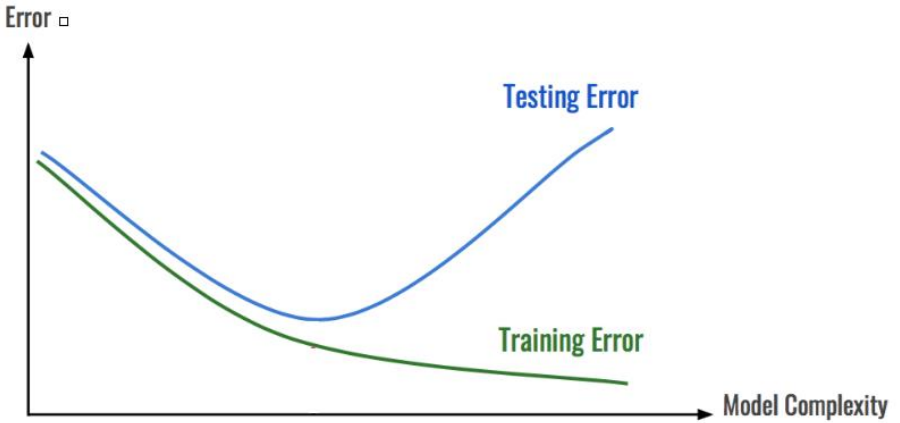
Overfitting

- You pick a "best" parameter set θ^*

Training Data: $\{\ldots(x^r, \hat{y}^r)\ldots\} \Rightarrow \forall r : f(x^r;\theta^*) = \hat{y}^r$

However,

Testing Data: $\{\ldots x^u \ldots\} \Rightarrow f(x^u;\theta^*) \neq \hat{y}^u$



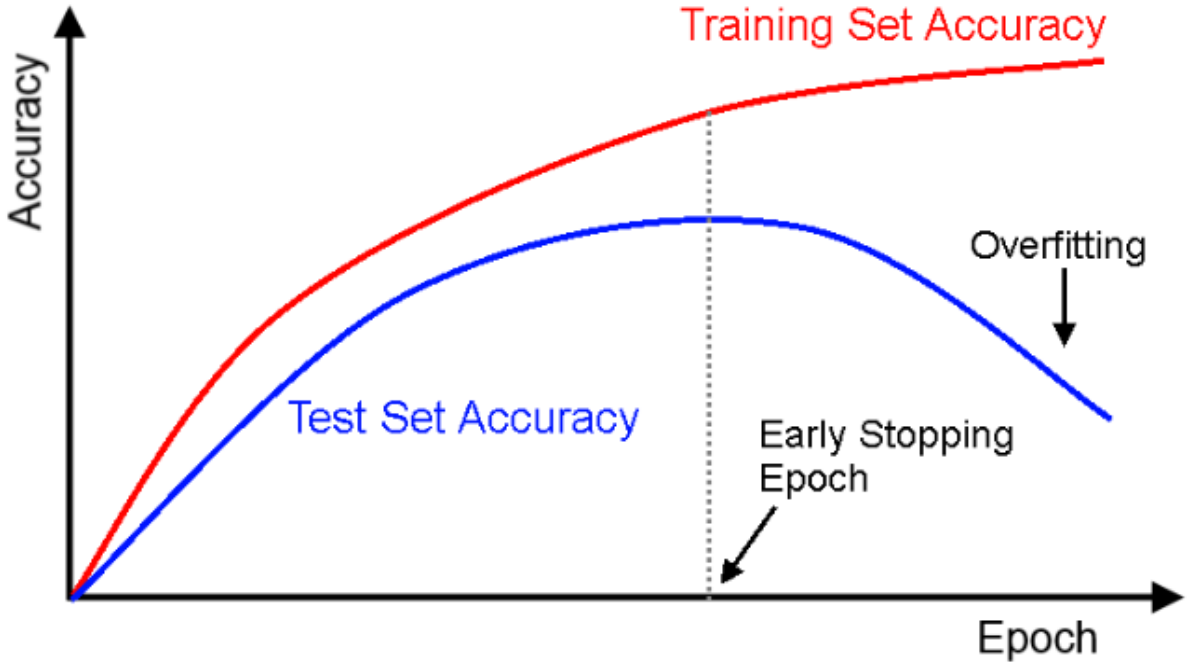
Training error is going lower but testing error is not.!!!

Prevent Overfitting

- Early stopping
- Stronger regularization
- Use more data
  - Data augmentation
- Reduce the complexity of model
  - Reduce Layer
  - Dropout

Early Stopping

- Evaluate the accuracy of validation data after every epoch.
- When the accuracy becomes stable, stop the training.
  - Solve set the number of epoch manually.
  - Solve overfitting problem.



Regularization

L1 Regularization

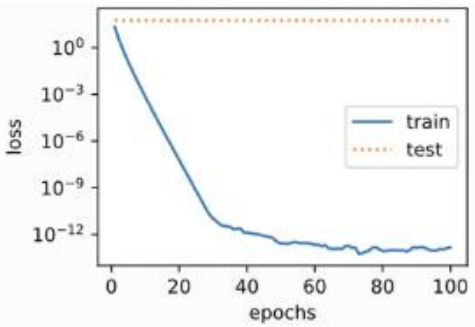
$$\text{Cost} = \sum_{i=0}^{N}(y_i - \sum_{j=0}^{M} x_{ij}W_j)^2 + \lambda \sum_{j=0}^{M} |W_j|$$

L2 Regularization

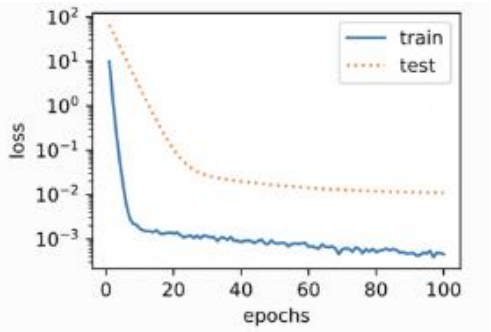
$$\text{Cost} = \sum_{i=0}^{N}(y_i - \sum_{j=0}^{M} x_{ij}W_j)^2 + \lambda \sum_{j=0}^{M} W_j^2$$

Loss function

Regularization Term



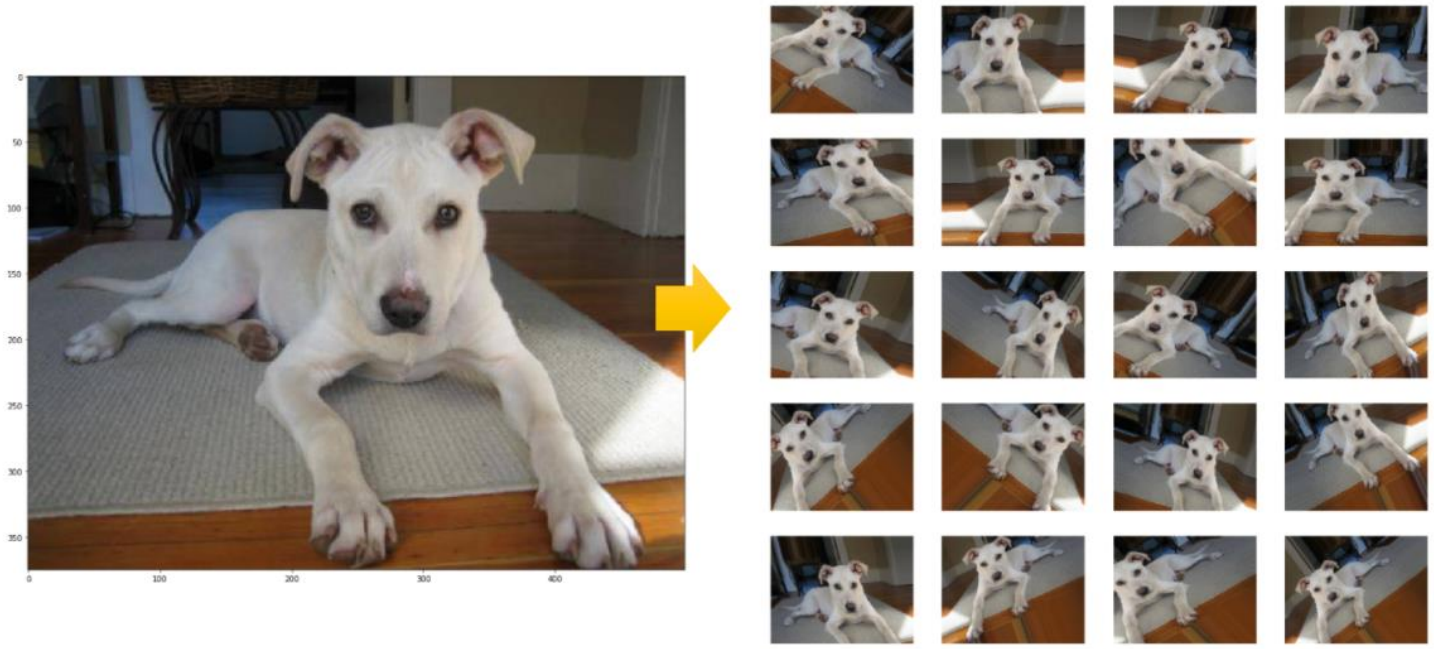
Overfitting



L2 norm

Data Augmentation

- Rotate, resize, change bright….etc. to increase the variance of data.
- Although human can differentiate is the same image, machine thinks they are different images.

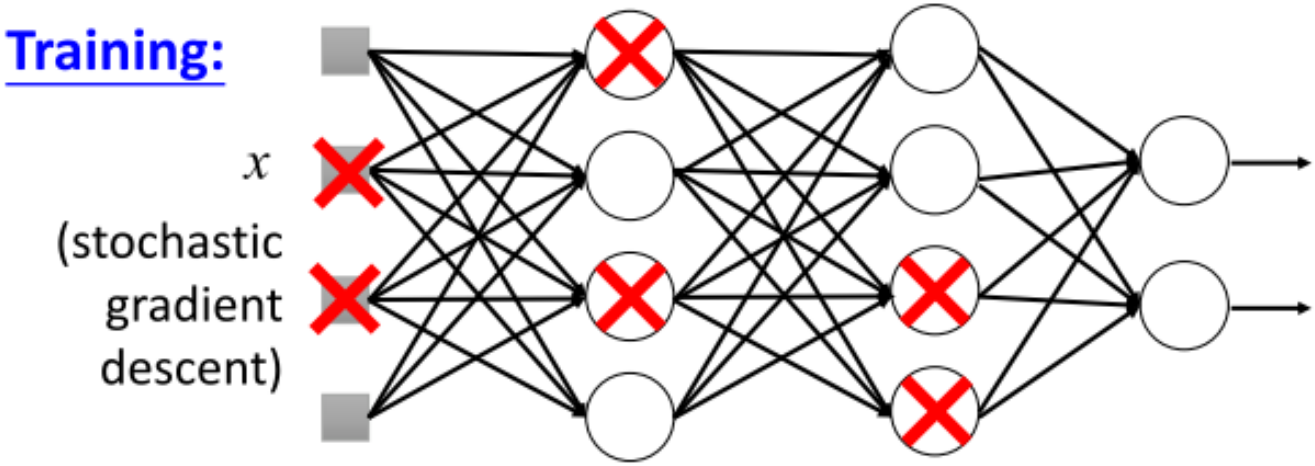


Reduce the Complexity of Model

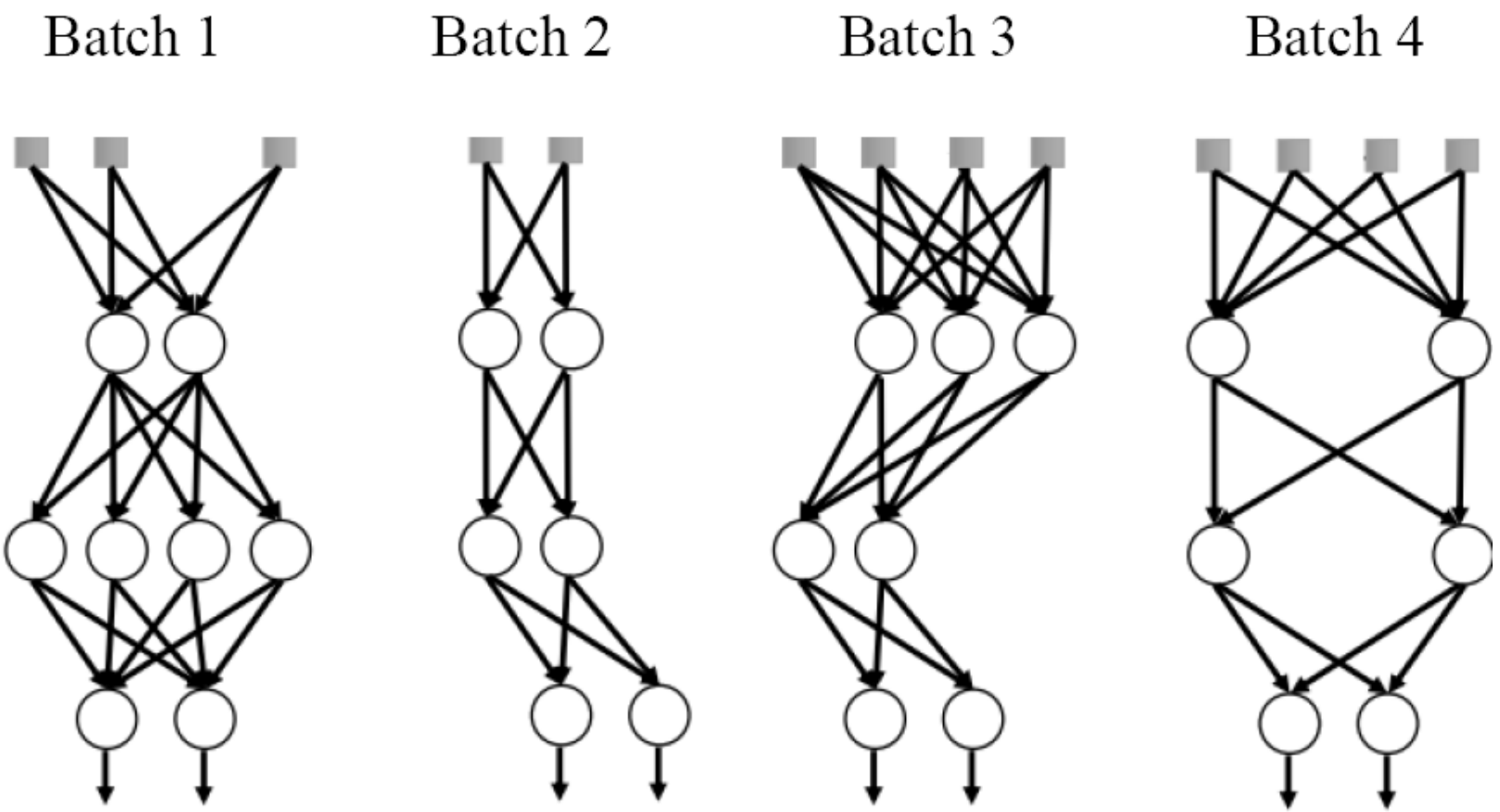
Dropout (1/2)

- Solve the problem of overfitting
- When training, each neuron has P% probability dropout.
  - Each mini-batch would resample the dropout neuron.

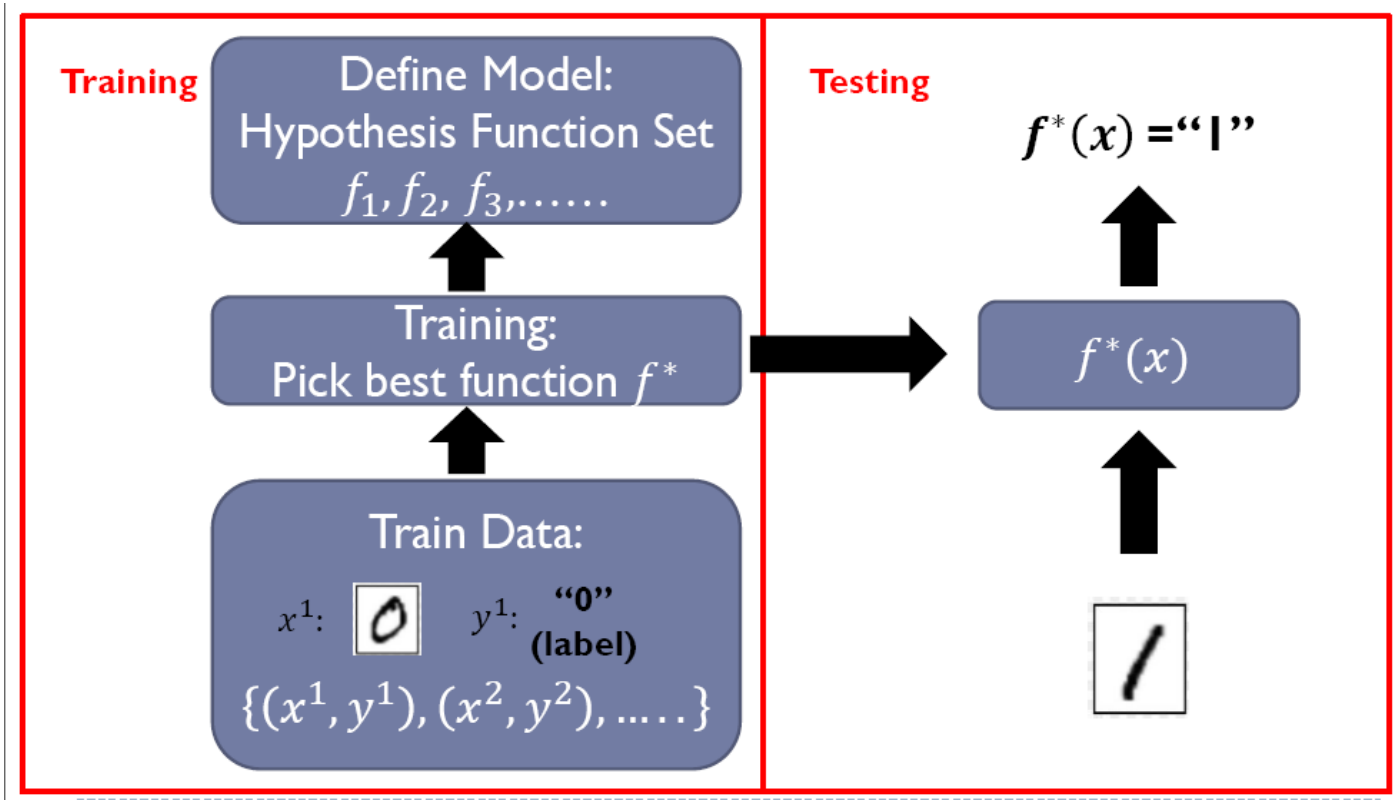
$$\theta^t \leftarrow \theta^{t-1} - \eta \nabla C_x(\theta^{t-1})$$



Dropout (2/2)



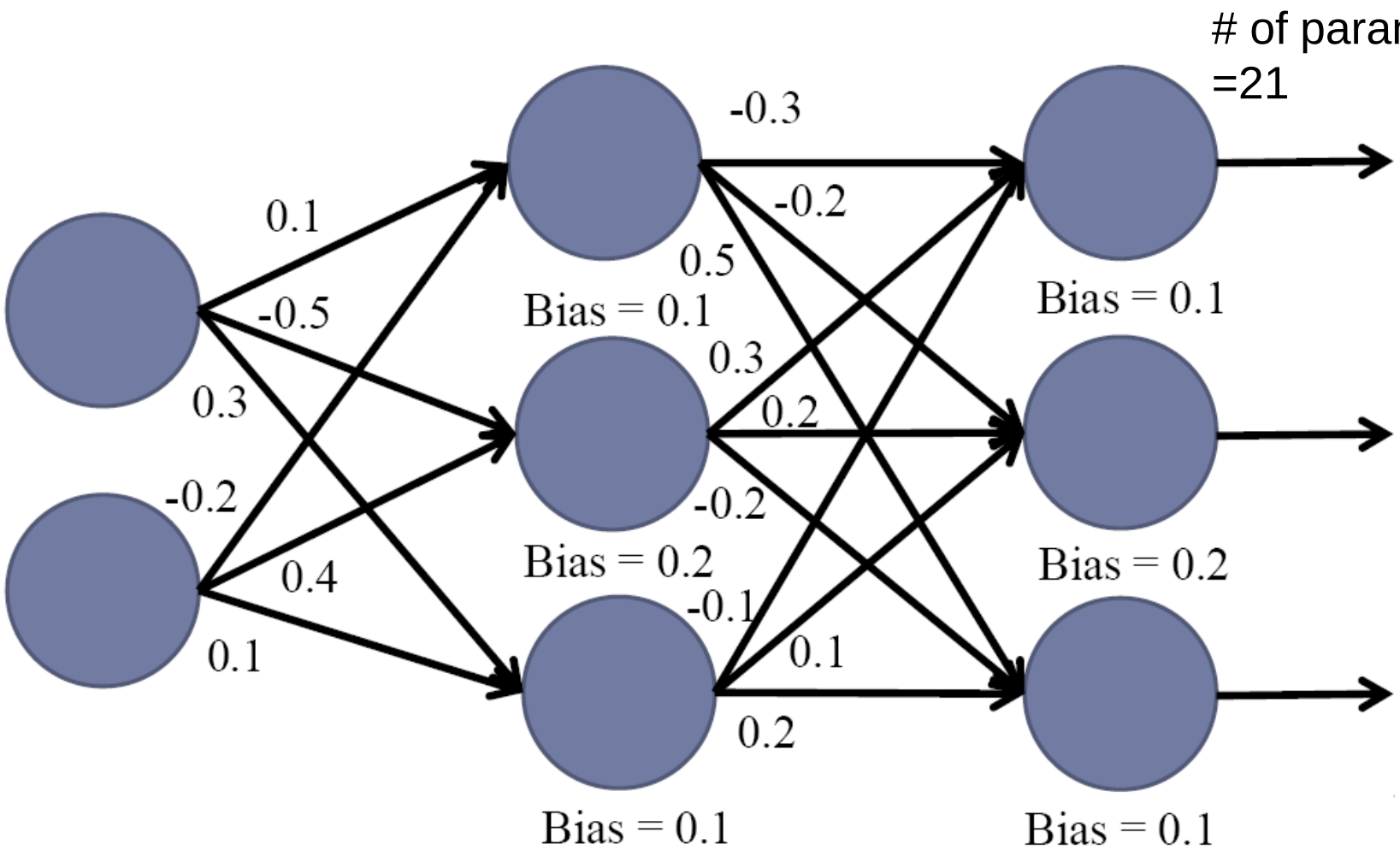
Summary:
What We Learn of DNN Model



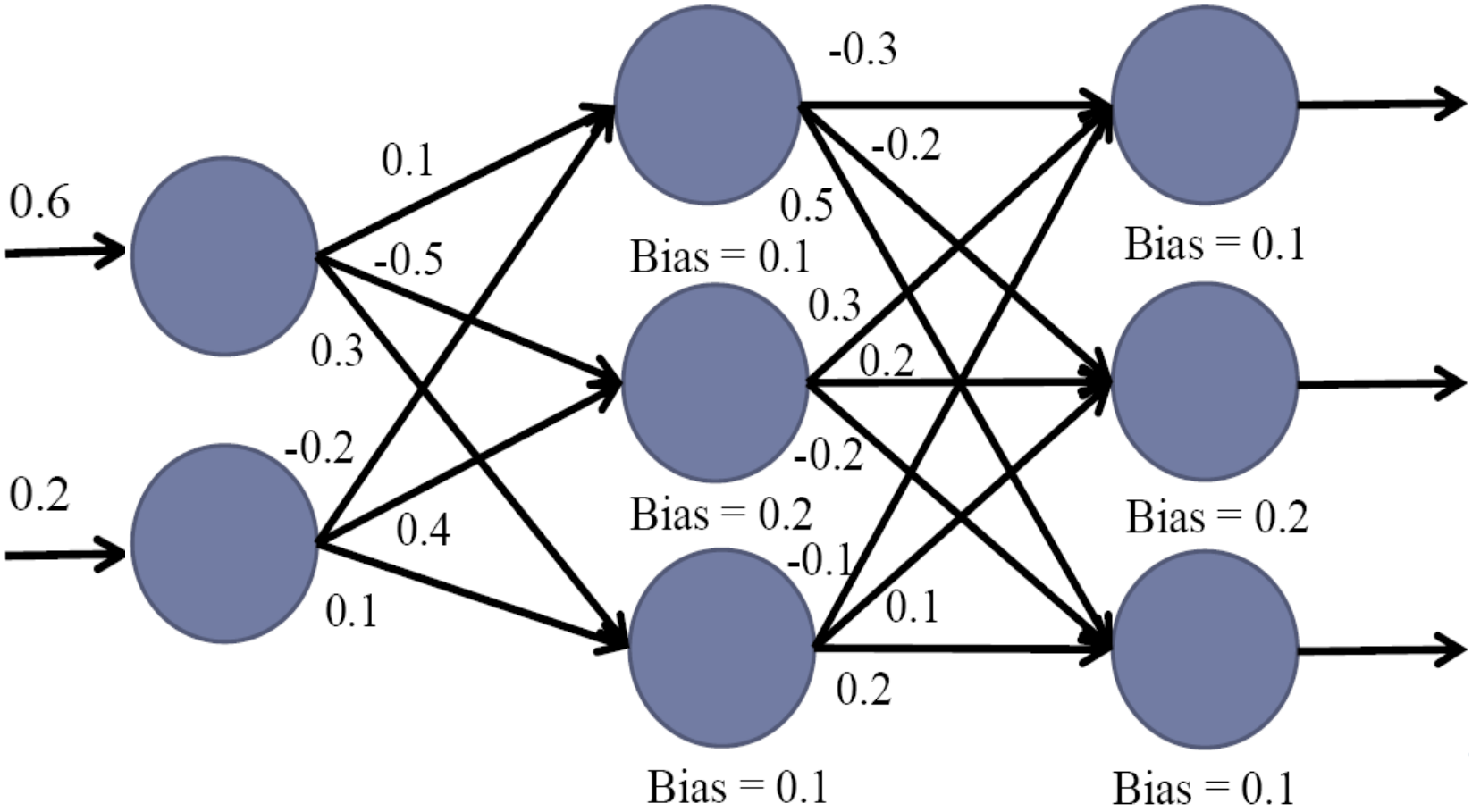
1. Prepare Dataset
2. Build DNN Model
3. Define Loss
4. Optimization

Predict/Validate Result

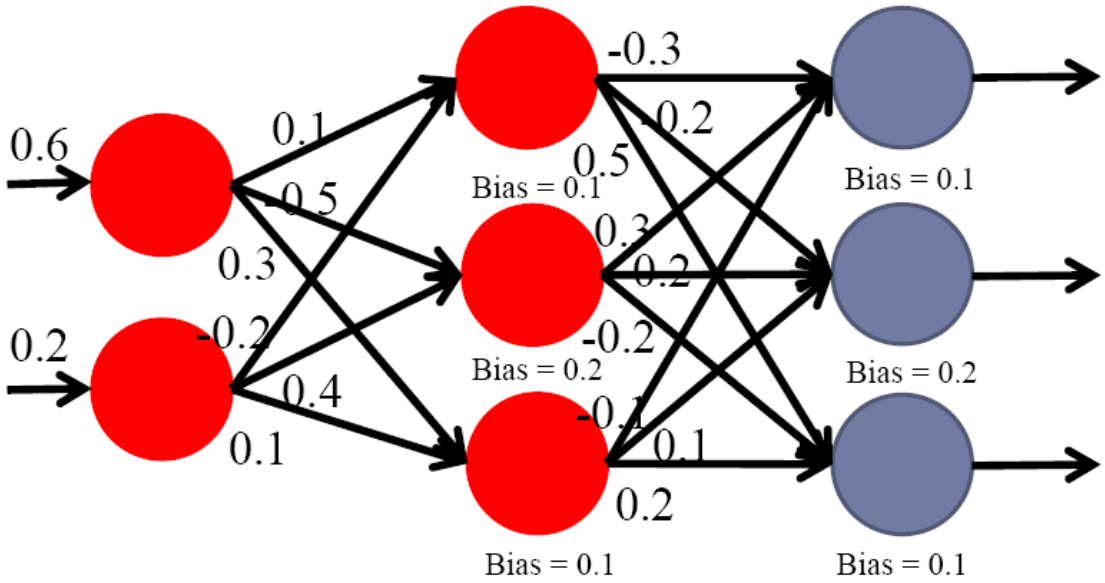
Example-Initialization



Example-Feed Data



Example-Forward Pass

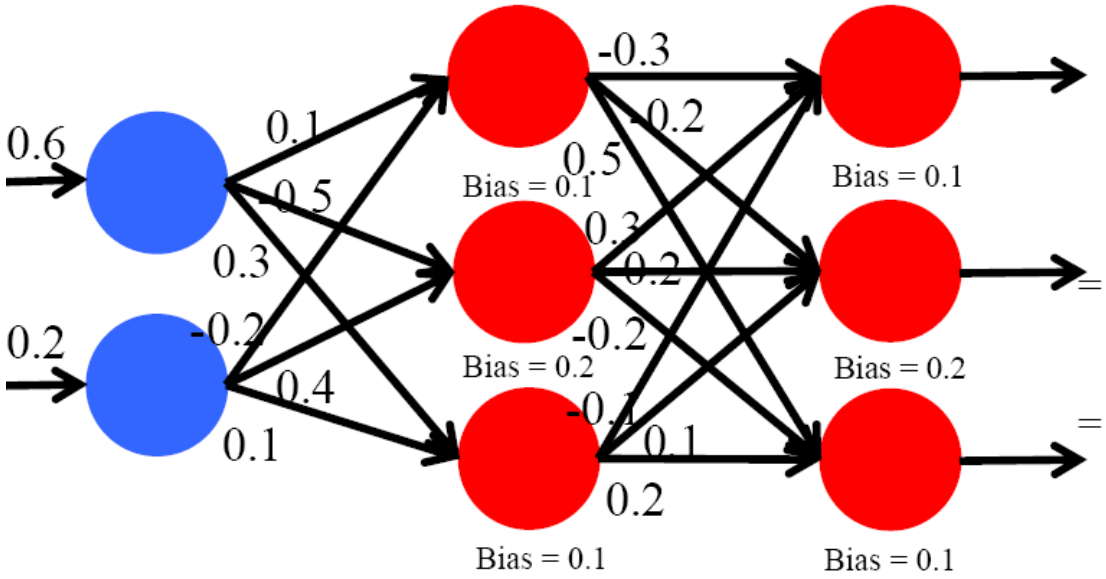


$$relu\left(\begin{bmatrix} 0.1 & -0.2 \\ -0.5 & 0.4 \\ 0.3 & 0.1 \end{bmatrix}\begin{bmatrix} 0.6 \\ 0.2 \end{bmatrix} + \begin{bmatrix} 0.1 \\ 0.2 \\ 0.1 \end{bmatrix}\right)$$

$$= relu\left(\begin{bmatrix} 0.12 \\ -0.02 \\ 0.3 \end{bmatrix}\right)$$

$$= \begin{bmatrix} 0.12 \\ 0 \\ 0.3 \end{bmatrix}$$

Example-Forward Pass

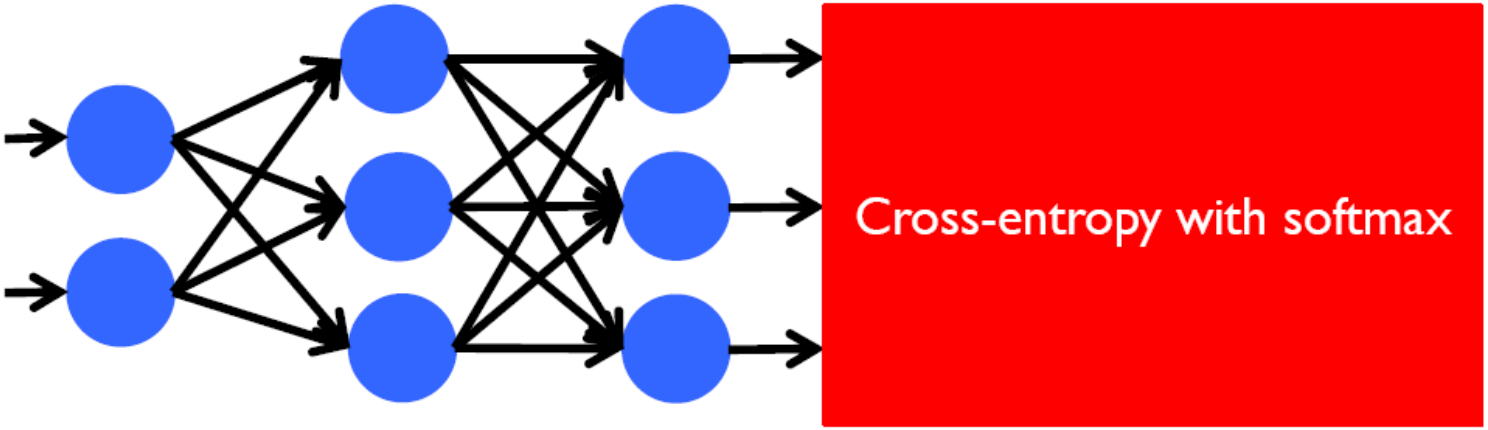


$$relu\left(\begin{bmatrix} -0.3 & 0.3 & -0.1 \\ -0.2 & 0.2 & 0.1 \\ 0.5 & -0.2 & 0.2 \end{bmatrix}\begin{bmatrix} 0.12 \\ 0 \\ 0.3 \end{bmatrix} + \begin{bmatrix} 0.1 \\ 0.2 \\ 0.1 \end{bmatrix}\right)$$

$$= relu\left(\begin{bmatrix} -0.066 \\ 0.006 \\ 0.12 \end{bmatrix}\right)$$

$$= \begin{bmatrix} 0 \\ 0.006 \\ 0.12 \end{bmatrix}$$

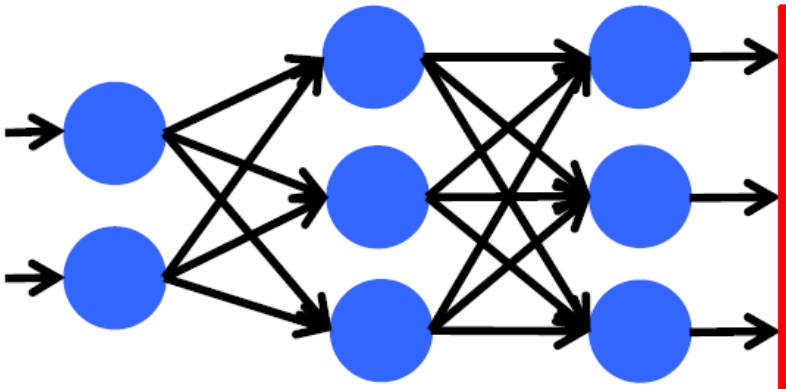
Example-Forward Pass



$$\text{Softmax}\left(\begin{bmatrix}0\\0.006\\0.12\end{bmatrix}\right) = \begin{bmatrix}0.319\\0.321\\0.36\end{bmatrix}$$

$$\sigma(\mathbf{z})_j = \frac{e^{z_j}}{\sum_{k=1}^{K} e^{z_k}} \quad \text{for } j = 1, \ldots, K.$$

Example-Forward Pass



Cross-entropy with softmax

What we expect (label)

$$\begin{bmatrix} 0 \\ 1 \\ 0 \end{bmatrix}$$

- 0 * ln(0.319) – 1 * ln(0.321) – 0 * ln(0.36)
= 1.1363

$$-\sum_{i=0}^{class\ \#} \hat{y}_i \ln(y_i)$$

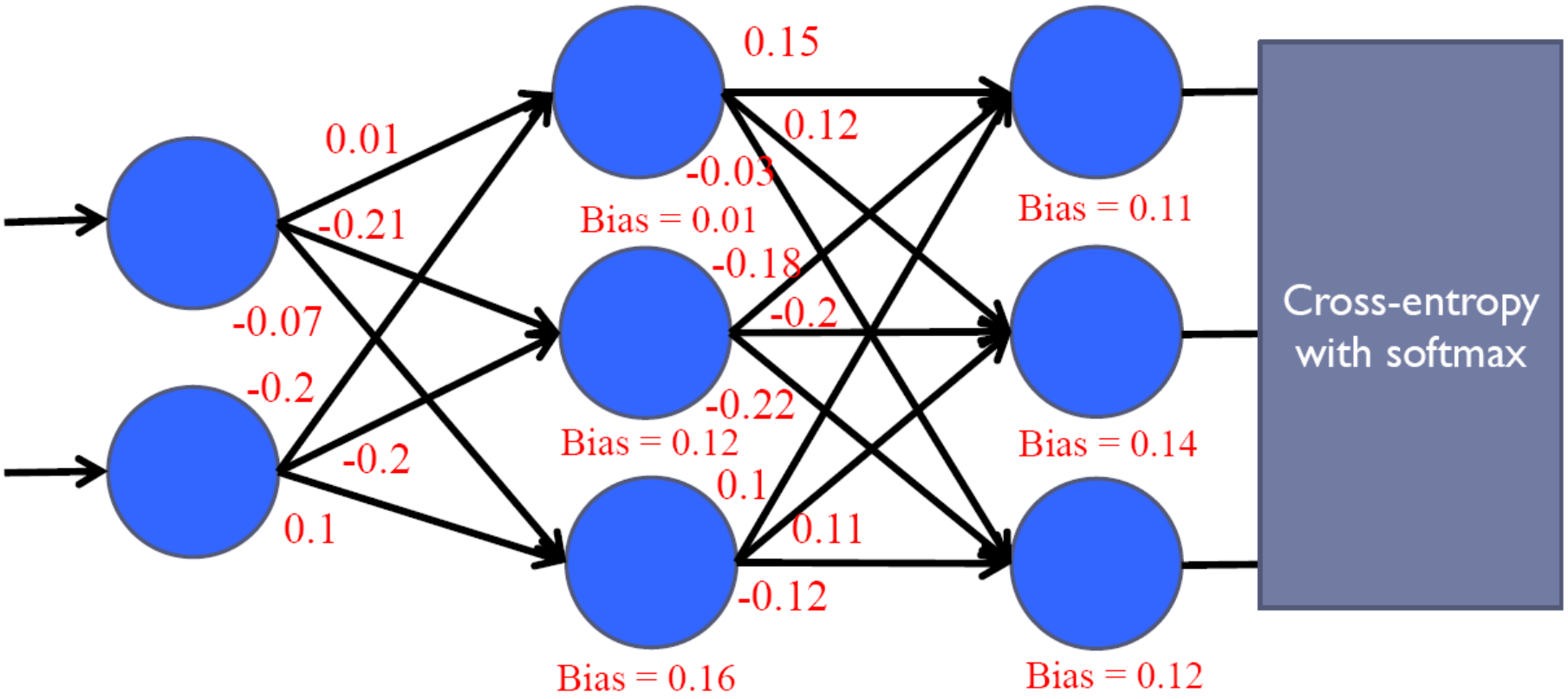
- This is large at the beginning. During training, this should decrease.

Example-Optimization

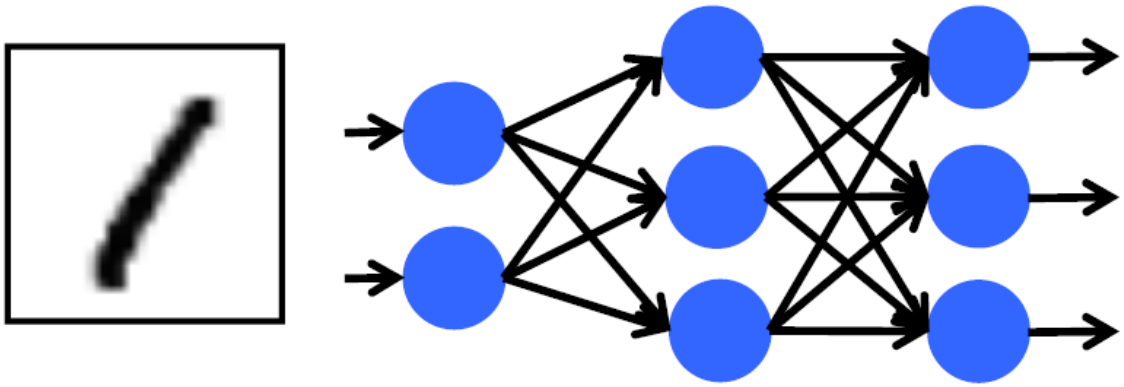
Use optimizer to optimize

and wait……

Example-After Optimization



Example-Predict



Feed you test data

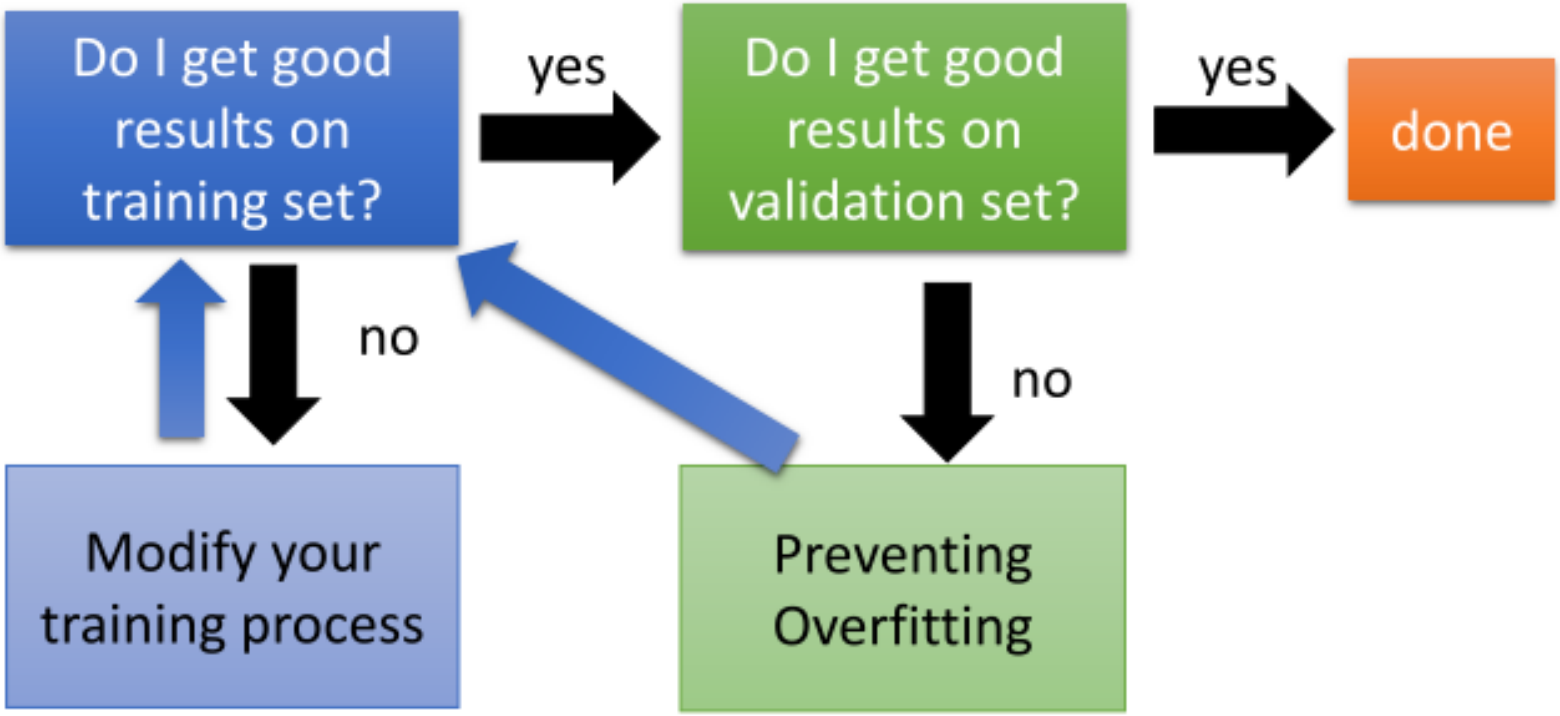
You may get something like this

$$\begin{bmatrix} 0.1 \\ 0.5 \\ 0.4 \end{bmatrix}$$

Model predict this is class #2

(0.5 is greater than others)

Recipe for Learning



- Your code usually do not have bug at this situation.